Translating Relational Databases into Linked Open Data

Master thesis for obtaining the Diploma Master in Computer Science and Engineering

Corentin Froment

Director
Professor Stijn Vansummeren

Co-Promotor
Alejandro Vaisman

Service
CoDE - WIT

Academic year
2011-2012
Abstract

With the growth of the Semantic Web, the existing paradigm of relational data for data storage becomes obsolete. Obtaining such data in a new, adequate format is impossible using manual method. Therefore, there is a strong need to create automated processes in charge of translating relational data into linked data.

In this thesis, we briefly remind the fundamental concepts necessary to its understanding. We shall develop the state of the art of this translation issue, presenting the current theories and implementations. We will end this thesis by presenting an implementation we have created to meet the requirements of two use cases.

Keywords  Semantic, Web, RDF, Relational Data, Translation, Linked Open Data, R2RML, Mapping, XML
Acknowledgements

This master thesis is the conclusion of my intense academic journey. Achieving it would no have been possible without the help, encouraging and monitoring of Stijn and Alejandro. Working with them has always been very stimulating and I have learned much more than the topic that thesis is dealing with.

I would like to thank each member of my family for having supported me during this long year, especially when work used to exhaust me too much. Thanks a lot Maman, Papa, Nicolas and Florence.

Of my life at the University, I will perhaps remember moments of study, but I will certainly remember the other moments of not study spent with my friends. Nicolas deserves a special attention for having read this thesis (and I do forgive the fact he found it boring, everyone can probably not be interested with the Semantic Web).

Finally, I thank François V. and all his brothers and sisters for making me laughing, crying and thinking, which was a great contribution for the completion of this part of my life.
Contents

1 Introduction .......................... 6
1.1 Context .......................... 6
1.2 Foundations of the Semantic Web .............. 6
  1.2.1 Unique Resources Identifiers .............. 7
  1.2.2 The Resource Description Framework ..... 9
  1.2.3 SPARQL .......................... 11
  1.2.4 The Linking Open Data project ......... 12
    1.2.4.1 DBpedia .......................... 13
1.3 The OSCB project and use cases .......... 15
  1.3.1 Agenda.be .......................... 15
  1.3.2 BOZAR .......................... 16
1.4 Thesis objectives and contributions ....... 17

2 State of the Art .................. 18
  2.1 Openlink Virtuoso Universal Server ...... 19
    2.1.1 Introduction .................. 19
    2.1.2 Conclusion .................. 22
  2.2 D2R Server .................. 22
    2.2.1 Introduction .................. 22
    2.2.2 Architecture & functioning ......... 22
    2.2.3 D2RQ mapping example .......... 23
    2.2.4 D2RQ Mapping Language ......... 24
      2.2.4.1 D2RQ Vocabulary .......... 24
    2.2.5 Summary .................. 25
  2.3 Direct Mapping .................. 26
    2.3.1 Introduction .................. 26
    2.3.2 Functioning .................. 27
      2.3.2.1 Simple Case .................. 27
      2.3.2.2 Link with other tables, referencing candidate keys .... 27
    2.3.3 Conclusion .................. 29
  2.4 R2RML .................. 30
    2.4.1 Introduction .................. 30
    2.4.2 Technical details .............. 30
      2.4.2.1 Global example of R2RML mapping ......... 30
      2.4.2.2 General overview .............. 32
      2.4.2.3 Logical tables .............. 33
      2.4.2.4 Term maps .............. 35
        2.4.2.4.1 Subject maps .............. 37
        2.4.2.4.2 Predicate-Object Maps ......... 37
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.2.4.3 Referencing other tables</td>
<td>37</td>
</tr>
<tr>
<td>2.5 db2triples</td>
<td>39</td>
</tr>
<tr>
<td>2.6 Triplify</td>
<td>39</td>
</tr>
<tr>
<td>2.6.1 Overview</td>
<td>39</td>
</tr>
<tr>
<td>2.6.2 Translation mechanisms</td>
<td>40</td>
</tr>
<tr>
<td>2.6.3 Summary</td>
<td>41</td>
</tr>
<tr>
<td>2.7 Comparative study</td>
<td>42</td>
</tr>
<tr>
<td>2.8 Conclusions</td>
<td>42</td>
</tr>
<tr>
<td>3 Use Cases and requirements</td>
<td>44</td>
</tr>
<tr>
<td>3.1 Use cases description</td>
<td>44</td>
</tr>
<tr>
<td>3.1.1 A use case: BOZAR</td>
<td>44</td>
</tr>
<tr>
<td>3.1.2 A use case: Agenda.be</td>
<td>45</td>
</tr>
<tr>
<td>3.2 Cultural and location ontology description</td>
<td>46</td>
</tr>
<tr>
<td>3.3 Requirements</td>
<td>47</td>
</tr>
<tr>
<td>4 Current Implementation</td>
<td>49</td>
</tr>
<tr>
<td>4.1 Implementation design &amp; architecture</td>
<td>49</td>
</tr>
<tr>
<td>4.2 R2RML extensions</td>
<td>50</td>
</tr>
<tr>
<td>4.2.1 XML sources translation</td>
<td>51</td>
</tr>
<tr>
<td>4.2.2 Semantized SQL tables</td>
<td>52</td>
</tr>
<tr>
<td>4.3 Results</td>
<td>54</td>
</tr>
<tr>
<td>4.3.1 Mapping to the ontologies</td>
<td>54</td>
</tr>
<tr>
<td>4.3.2 Ontologies enrichment</td>
<td>57</td>
</tr>
<tr>
<td>4.3.3 Translation of the relational data</td>
<td>59</td>
</tr>
<tr>
<td>4.3.4 Performance analysis</td>
<td>60</td>
</tr>
<tr>
<td>5 Conclusion</td>
<td>63</td>
</tr>
<tr>
<td>A Introduction: BOZAR tables</td>
<td>66</td>
</tr>
<tr>
<td>B State of the Art: Results of the running example mappings</td>
<td>68</td>
</tr>
<tr>
<td>C Implementation architecture</td>
<td>72</td>
</tr>
<tr>
<td>D BOZAR mapping</td>
<td>73</td>
</tr>
<tr>
<td>E Agenda.be mappings</td>
<td>80</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 The Web of Data principle ........................................ 7
1.2 Venn diagram of the URIs .......................................... 8
1.3 Introduction to RDF .................................................. 10
1.4 Blank node example .................................................. 10
1.5 Linking Open Data global graph, September 2008 .............. 13
1.6 Linking Open Data global graph, September 2011 ............. 14
1.7 The OSCB project ..................................................... 16

2.1 OpenLink Virtuoso Universal Server architecture ............... 20
2.2 D2RQ Server architecture ........................................... 23
2.3 D2RQ Mapping structure example .................................. 26
2.4 Example of simple direct mapping .................................. 28
2.5 Properties of logical tables ......................................... 34
2.6 Term maps properties ............................................... 35
2.7 Triplify translation mechanism ...................................... 40

4.1 GUI ........................................................................ 50
4.2 Transformation of a semantized table .............................. 53
4.3 Logical table building times ......................................... 61
4.4 Speed of translation .................................................. 61
4.5 Total translation time .................................................. 62
## Listings

1.1 Resulting triples of the simple RDF graph | 1.3 .............................. 10
1.2 RDF : XML/RDF notation .............................. 10
1.3 RDF : Notation 3 (N3) .............................. 11
1.4 RDF : Turtle notation .............................. 11
1.5 SPARQL Query example .............................. 12

2.1 OpenLink Virtuoso, RDB2RDF example .............................. 20
2.2 Example of D2RQ Mapping .............................. 23
2.3 Result of a simple Direct Mapping .............................. 27
2.4 Result of the application of Direct Mapping .............................. 28
2.5 Direct Mapping on tables with special foreign keys .............................. 28
2.6 R2RML global mapping example .............................. 30
2.7 Mapping syntax, example 1 .............................. 32
2.8 Mapping syntax, example 2 .............................. 32
2.9 Logical table, rr:tableName .............................. 34
2.10 Logical table, rr:sqlQuery & rr:sqlVersion .............................. 34
2.11 Constant-valued term map .............................. 36
2.12 Column-valued term map .............................. 36
2.13 Template-valued term map .............................. 36
2.14 Equivalent SQL query for a rr:joinCondition .............................. 38
2.15 Join Condition example .............................. 38
2.16 PHP example of triplify ‘mapping’ .............................. 41

3.1 Overview of the Agenda.be data structure .............................. 45
3.2 Insight of the GOSPL cultural ontology .............................. 46

4.1 Simple XML document .............................. 51
4.2 R2RML predicate-object map for generating event titles in BOZAR .............................. 56
4.3 R2RML predicate-object map for generating event titles in Agenda.be .............................. 56
4.4 Logical table for BOZAR activity table .............................. 56
4.5 Resulting triples of the BOZAR translation .............................. 59
4.6 Slice of an event node of the original ‘event’ XML file .............................. 59
4.7 Partial result of translation of the agenda.be event database .............................. 60

B.1 Virtuoso RDF View of the running example .............................. 68
B.2 D2RQ mapping results .............................. 69
B.3 RDF dump of R2RML mapping applied on the running example .............................. 70

D.1 BOZAR mapping .............................. 73

E.1 Agenda.be events mapping .............................. 80
Chapter 1

Introduction

In this chapter, we will introduce the Master Thesis by describing the context in which it is taking part. Then, we will remind the foundations of the Semantic Web and the Linking Open Data project. We will present the OSCB project and quickly introduce the use cases. This will lead us to deal with the actual thesis objectives.

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Context</td>
<td>6</td>
</tr>
<tr>
<td>1.2 Foundations of the Semantic Web</td>
<td>6</td>
</tr>
<tr>
<td>1.3 The OSCB project and use cases</td>
<td>15</td>
</tr>
<tr>
<td>1.4 Thesis objectives and contributions</td>
<td>17</td>
</tr>
</tbody>
</table>

1.1 Context

Nowadays, the majority of the data stored on the Internet is often in a relational form, mainly in relational databases (SQL, Oracle, ...). With the rise of the Semantic Web, there is need of having data into RDF. However, recreating these data manually is infeasible. This statement implies the necessity to find automatic processes for converting relational data into RDF.

Before diving any further in the description of those processes, we provide next the very minimal knowledge needed to understand the thesis, namely an introduction to the semantic Web, a reminder about URIs and RDF and an overview of the Linking Open Data project.

1.2 Foundations of the Semantic Web

Most of the Web content’s today is designed for humans to read, not for computers to manipulate \[13\]. Of course computers can process some documents (e.g. HTML documents with headers, links, ...) but in general they are unable to understand the semantics behind those documents. The goal of the Semantic Web is to meaningfully structure the content of the Web. It is not a new Web, but rather an extension of the existing one.

Instead of having a bunch of unstructured documents, the idea of is to have bunch of structured pure data. The proposed structure can be seen as a giant graph where data are interconnected with each other, as shown in the figure below:
These data can hence be easily manipulated by computers, since the Semantic Web gets rid of the superfluous data that was necessary for the human read. Furthermore, it is possible to define ontologies specifying the links that can or cannot exists between certain type of data\[17\]. Once these links defined, computer have the possibility to infer knowledge which was not specified by humans. The Semantic Web offers hence a formidable opportunity for the evolution of human knowledge.

A good approach to understand the two base concepts of the Semantic Web is to ask two questions:

- How do we represent a thing in the Semantic Web ?
- How do we link the different things ?

1.2.1 Unique Resources Identifiers

To answer the first question, we have to keep in mind that everything can be represented. Instead of talking about things, we will prefer the term of resources.

“Any information that can be named can be a resource: a document or image, a temporal service (e.g., “today’s weather in Los Angeles”), a collection of other resources, a nonvirtual object (e.g., a person), a concept and so on.” \[18\]

So, almost everything that exists can be represented. However, it should be good to be able to imagine a representation with no ambiguity. For instance, someone could be the resource ‘John Doe’, but his neighbour, who could want to be a resource as well can have the exactly
same name, and would be represented as ‘John Doe’. We must hence use a system that uniquely identifies a resource. This is the role of the *Unique Resource Identifiers* (URIs).

![Venn diagram of URIs](image)

**Figure 1.2:** Venn diagram of the URIs, containing URLs, URNs and their intersection

As we can see on the Venn diagram in Fig 1.2, a URI can be either a URL, which stands for *Unique Resource Location*, or URN, that is *Unique Resource Name* or both.

If we look at the structure of a URI, we can see that it must take the following form:

```
scheme:scheme-specific-part
```

The scheme part, which ends with the colon character ‘:’, consists in a sequence of letters, digits, plus (‘+’), period (‘.’) and hyphen (‘-’); that sequence must begin with a letter. Most of the time, the scheme part refers to a protocol, as `http` or `ftp`, since most URIs were originally created to be used with a particular protocol. In our case, the scheme part can be anything we want, since URIs are meant to be used as identifiers.

The scheme specific part can be divided in three part:

```
<hierarchical part> [ ? <query> ] [ # <fragment> ]
```

The hierarchical part provides information about the hierarchy of a resource (a path, for instance). This part is subdivided in an authority and a path. The authority may contain user information, a host name and a specific port. Note that the authority or the path can be omitted.

The query is the part following a question mark. There is no particular syntax describing a query, but usually, we will find a sequence of ‘key=value’ pairs separated by semi-colons, ampersand (&) or whatsoever a bit particular.

Finally, the fragment part, that follows the sharp character, provides additional information about another resource, for example a section or a paragraph in an HTML web page.

[^1]: http://tools.ietf.org/html/rfc3986#appendix-A
The syntax that is described here above applies for both URLs and URNs. The difference between these two kind of identifiers lies in their meaning. A URL will give indication about the location of a resource. If we take the former example of a person considered as a resource, a URL would give the address of this person and we would know where he is standing. A URN will give information about the name of a resource and therefore we would know exactly what (or who) it is.

The following are all examples of URIs:

- http://localhost:8080/example/uri_1?type=section
- http://www.w3.org/1999/02/22-rdf-syntax-ns#type
- urn:isbn:0-395-36341-1
- ftp://johndoe:s3cr3tP4ss@example.com:21/home/jdoe/resource.txt

The third example shows very well the difference between a URL and a URN. The given ISBN clearly identifies the book (there is no other book with the same id) but does not give any information about the place where it could be bought. In a same way, the other examples are giving information about location, but it is impossible to know what stands on those very places.

As a last word about URIs, we can mention the existence of IRIs, that stand for Internationalized Resource Identifiers, which are a superclass of URIs (every URI is an IRI but the opposite is not true). The principle is the same, that is, the goal is to unequivocally identify a resource, but IRIs are not limited to the ASCII encoding, which is the case of URIs. However, every IRI can be translated in URI by replacing special characters (such as a 'ø') with an ASCII sequence ('ø' becomes '%F8').

1.2.2 The Resource Description Framework

The second (and probably most important) brick constituting the foundation of the Semantic Web is the Resource Description Framework, RDF. RDF is a model to represent data and their different properties. To do so, RDF uses the formalism of URIs. The easiest way to understand RDF is to imagine the data as a graph in which every element, whether it is a node or an edge, is either a URI or a literal.

As a simple example, let us take a person whose we want his age and gender to be represented. The corresponding RDF graph is shown in Fig 1.3.

The two figures above represent the same thing, a female person of twenty four. There are three important parts in a graph : the “start point” (i.e., the Person), the link (i.e., the age and the gender) and the “end point” (i.e., 24 and 'female). In RDF, the first one is called the subject, the link is named predicate and the the last part the object. It is impossible to find a subject with a predicate without object or a standalone object. Such a relation is called a RDF triple (or RDF statement) and an RDF graph is a set of RDF triples.

Subjects and objects elements are named nodes. Nodes can be URIs (absolute URIs), literals or blank nodes. A blank node is just a node that has no name and which cannot be a URI or a literal.

http://www.w3schools.com/tags/ref_urlencode.asp
http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/#section-URI-Vocabulary
Figure 1.3: First example of the RDF representation, in a conceptual way at left and in a correct way at right.

literal. However, it may be used in one or more RDF statements. Here is an example of blank node:

Figure 1.4: Simple example of a blank node; predicates may take any admissible value

Although, if the graphical representation clearly explains the mechanisms of RDF, it is not the one used to store the triples. In general, RDF triples appear rather under the following textual form, which comprises all the information founded in the graphs presented in Fig. 1.3:


Listing 1.1: Resulting triples of the simple RDF graph

There are various formats in which RDF graphs can be expressed. One of the possibilities is to use XML. The main issue with the RDF/XML serialization is that it is not really human-readable. To solve this problem, representations of other kind have appeared, namely turtle (.ttl files) and n3 (n-triple). Here is a simple example with the three types and two statements:

```
## XML/RDF SYNTAX
< rdf : RDF
  xmlns : rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns : foaf ="http://xmlns.com/foaf/1.0/">
  < rdf : Description rdf : about ="http://thesis.example.org/JohnDoe">
    < foaf : name > John Doe </ foaf : name >
    < foaf : age > 24 </ foaf : age >
  </ rdf : Description >
</ rdf : RDF>
```

Listing 1.2: RDF : XML/RDF notation
Clearly, both last examples are easier to read.

Previously we said that all the resources could be identified through URIs. This allows everyone to create his own URIs defining its own concept. For instance, we could define the concept of master thesis at http://ulb.ac.be/semantics#master-thesis. In the same way, we could have defined the concept of ‘Person’ at http://ulb.ac.be/semantics#Person. However, it is considered as a good practice to avoid multiplying already existing concepts, since the final goal of semantic data is being understandable. A wide proliferation of identical semantics (such as ‘Person’) hugely complicates the task. That is why there are some “recommended” namespaces that precisely define concept used on a daily basis. We can name, for example, FOAF, RDF,
RDFSchemef...

The possibilities offered by RDF are extremely large. It would be sad not to mention ontologies. Ontologies are structured set of data that give meaning and definition about concepts. They allow the mechanism of inference with semantic data. For example, it is because an ontology defines the concept grandfather that the computer can deduce that the child of one’s child is his grandchild.

To use ontologies, the Web Ontology Language (OWL) has been created OWL is based on the RDF, which means that an OWL document is an RDF graph than can be processed as any other graph (OWL is to RDF what XML Schema is to XML). In this thesis, a big part of the duty was to create and use an ontology defining the concepts of the several use cases (see Sec. 1.3).

1.2.3 SPARQL

In our story so far, we have data which is published into RDF. However, the purpose of this data is to be manipulated and queried. To do so, the W3C has developed the SPARQL standard. SPARQL stands for SPARQL Protocol and RDF Query Language. It is for RDF...
what SQL is for relation data. It allows to create, search for, modify or delete data in RDF graphs.

SPARQL itself uses RDF. Most of the SPARQL queries will contain a set of triple patterns called basic graph pattern [3]. Triple patterns very similar to RDF triples except that each part of the triple (subject, predicate, object) can be a variable.

```sparql
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

SELECT ?age, ?name
WHERE {
  ?person foaf:gender 'female'.
  ?person foaf:name ?name.
}
```

Listing 1.5: SPARQL Query example

In the previous example, we have an example of a simple SPARQL query that illustrates the query mechanism. Let us picture an RDF graph similar composed of plenty of persons like the one we presented in Fig. 1.3b. The query would give us the age and the name (SELECT clause) of all the objects of kind foaf:Person (first where clause) who are females (second clause) and where name and age are objects associated to the predicates foaf:name and foaf:age (third and fourth clauses). All the terms preceded by a question marks are the variables of the query. The SPARQL engine is in charge to find the RDF data that matches the different triple patterns.

The power of SPARQL lies in the fact that one can write patterns matching literals, resources, languages tags, datatypes,... A pattern may contain more than one variable as well (e.g., ?s ?p 42 will match all the triples having the integer 42 as object).

1.2.4 The Linking Open Data project

The concept of Linked Data appeared with the rise of Web of Data, equally known as Semantic Web or Web 3.0. It is actually the foundation of the Web of Data. The main principles of Linked Data were initially pointed out by Tim Berners Lee, Web founder, in 2006 [11] and widely popularized in 2009 at the Technology Entertainment Design conference.

The main principles are the followings [13]:

1. Use IRIs as name for things
2. Use HTTP IRIs, so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs, so that they can discover more things.
The Web of data can be accessed thanks to Linked Data browsers, just as Web of Documents, composed of interlinked HTML documents can be accessed with HTML browsers. Though, rather than navigating between documents, the user can now indefinitely follow the link between actual data. The links between data are made with RDF (see Sec. 1.2.2). As we have seen previously, RDF links themselves are data. That allows us to state that if a there exists a link between two persons then those two people know each other.

In 2007, a project called the Linking Open Data project began publishing and interlinking many freely available datasets (encyclopaedias like Wikipedia, open government data, scientific datasets) in RDF using the above-mentioned Linked Data principles.

The Linking Open Data project is undoubtedly responsible of rise of the Web of Data recognition. At the end of 2007, there were already two billions of triples available. At this date, i.e., five year later, the number of triples is estimated at 31 billion.

The set of triples that belong to the Linking Open Data project can be represented in a big graph, a cloud, where nodes are datasets from different projects. In 2008, the cloud counted already around thirty distinct interlinked datasets, as shown in Fig. 1.5.

Three years later, the cloud grew significantly, as it appears in Fig. 1.6. This picture clearly shows the enthusiasm of the community for the project. If it continues to progress at the same rhythm, it is probably the last year that is possible to show the cloud on paper.

1.2.4.1 DBpedia

In both figures picturing the linked open data cloud, the central dataset is DBPedia, a project aimed at retrieving structured information from the famous free online encyclopaedia Wikipedia.
Figure 1.6: Linking Open Data global graph, September 2011. The red circles are the datasets that were already present in 2008.
Most of the people is familiar with the layout of this platform, where relevant information and data are presented in infoboxes on the top of the page. This information is structured in a formatted table based on templates, which means that is possible to extract data directly and interpret it.

DBpedia retrieve the data in Wikipedia’s infoboxes and publish them in RDF following the Linking Data fundamental concepts. DBpedia provides a SPARQL endpoint to query the triples. Moreover, it is very easy to interlink other datasets to DBpedia with RDF links. The datasets are thus fairly easy to access since DBpedia is often the first queried platform.

1.3 The OSCB project and use cases

This thesis is part of the OSCB project (Open Semantic Cloud for Brussels). OSCB has started in 2011 and is planned to end in 2014. The goal of the project is to create a platform, metaphorically named cloud, for Linked Open Data for covering region of Brussels. OSCB can be seen as a future node in the semantic datasets cloud of the Linking Open Data Project.

The project is aimed around four development axis:

- The creation of an ontology allowing the gathering of different actors in Brussels (e.g., cultural actors).
- The automation for annotation of images and videos
- The atomization of databases
- A multilingual terminology

As we will detail later in this report, the thesis takes part in two of those projects, namely the atomization of databases and the creation of the ontology.

The OSCB justifies the use of linked open data to provide a uniform access to public data, encouraging hence the reuse of the data to create new applications and use cases and stimulate the innovation.

The goal of the project is to lead as much actors in the everyday-life in Brussels as possible to carry their data into RDF (the framework used to express linked data) so they can easily share and reuse them, as shown in Fig. 1.7 Several industrial actors are already involved in the project: VRT, RTBF, Tour&Taxis, Actiris, ... In this thesis, we have two institutions participating in the cultural life in Brussels as use cases, Agenda.be and BOZAR. The objective of the thesis is to provide a solution allowing those actors to participate in the OSCB.

1.3.1 Agenda.be

Agenda.be is an organization whose role is the coordination of the maintenance of the cultural agenda in Brussels particularly and in Belgium in a lesser degree. It publishes its data in XML. Each time there is new data (for instance every week), new XML exports are created. A more accurate description of the compositions of those XML files is made in Sec. 3.

We are in possession of two those files. The first one is a compilation of all the events that have already been organized. The second one deals with the different institutions recorded by Agenda.be.
Those XML files can be considered as relational data, the different tuples being siblings, children of the same root node. Each of them has the same children element nodes, whose names can be considered as the different values of the schema and whose text nodes can be considered as values in the tuples.

Agenda.be is an aggregation of data published by other cultural institutions. Each time they publish new data, these must be re-encoded by Agenda.be in their own format, with an eventual loss of information if information cannot be converted. Using linked data as common data format would solve this problem since linked data can be directly reused and hence perfectly justifies the participation of Agenda.be to the OSCB.

1.3.2 BOZAR

BOZAR is a Belgian cultural institution covering music, expositions, cinema, theatre, dance, literature and so on. Inaugurated in 1928⁸ it is therefore a major actor in the Belgian cultural event organisation nowadays.

BOZAR stores its data into a SQL database. The list of the different table can be found in annex of this report (Annex [A]). The architecture of the database may already be pointed out; the database is conceptually divided in two parts: the tables describing things (activities, opuses, ...) and those linking the data. The last ones have a name that begins with an underscore character (‘_’).

By participating in the OSCB project, BOZAR can highly ease the publishing of its data to other institutions. For now, each time that new data is encoded, the entire database must be dumped and published. By using linked data, BOZAR would have only to provide the changes.

⁸http://en.wikipedia.org/wiki/Centre_for_Fine_Arts,_Brussels
Moreover, the reuse of linked data would be easier for the other institutions as well since the database structure of BOZAR is complex.

1.4 Thesis objectives and contributions

Nowadays, the majority of the data stored on the Internet is often in a relational form, mainly in relational databases (SQL, Oracle, ...). With the rise of the semantic Web, there is need of having data into RDF. However, recreating these data manually is infeasible. There is hence a clear need for translating relational databases into RDF.

To respond to this demand, several theoretical frameworks and proposals of implementation have already been made. The first objective of this thesis will be to give an overview of these different techniques by presenting their functioning and comparing their respective strengths and weaknesses.

Providing a better data management system for the region of Brussels by participating in the OSCB project is what motivated us to undertake such a thesis. To do so, converting data from in production/available relational databases at use in various cultural actors in Brussels into RDF is unaffordable.

However, to make this translation possible, the current state of the art is not sufficient. Hence, a certain of extensions to the state of the art are is necessary.

To fulfil these different requirements, we have implemented several extensions and we illustrate this solution by providing mapping of the relational databases of the actors, namely BOZAR and Agenda.be, to RDF.
Chapter 2

State of the Art

In this chapter we will present the current State of the Art of the methods and paradigms concerning the translation of relational data into linked data. We shall first present some implementations like D2R server and Openlink Virtuoso Server among others before exploring in details the two standards developed by the W3C. To conclude this chapter, we will propose a comparative study of the different solutions of the State of the Art.

Contents

2.1 Openlink Virtuoso Universal Server .................................. 19
2.2 D2R Server ........................................................................ 22
2.3 Direct Mapping .................................................................... 26
2.4 R2RML ............................................................................... 30
2.5 db2triples .......................................................................... 39
2.6 Triplify ............................................................................... 39
2.7 Comparative study ............................................................... 42
2.8 Conclusions ...................................................................... 42

Preamble - running example

We will illustrate the functionalities of existing database-to-RDF technologies by means of the following running example throughout this section.

Let us consider the following tables and table schemas. Values are given to compare them with the results in RDF given in Annex B.

<table>
<thead>
<tr>
<th>DATABASE</th>
<th>TABLENAME</th>
<th>SCHEMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIVERSITY</td>
<td>LABORATORIES</td>
<td>LID, LNAME, LLOCATION, HEAD_ID</td>
</tr>
<tr>
<td>UNIVERSITY</td>
<td>RESEARCHERS</td>
<td>RID, RNAME, LAB_ID</td>
</tr>
</tbody>
</table>

In the example, we will consider a database of a fictitious university and two of its tables. The first one contains a list of the different laboratories, identified by and id (LID) and having a name (LNAME), a location (LOCATION) and director (HEAD_ID), who is referencing an entry in the RESEARCHERS table (ref RID).
The other table contains the different researchers of the university. Each is described by a unique id (RID), a name (RNAME) and is attached to a laboratory (LAB_ID), which corresponds to an entry in the LABORATORIES table. We will assume that each laboratory director is also a researcher of this laboratory.

<table>
<thead>
<tr>
<th>LID</th>
<th>LNAME</th>
<th>LOCATION</th>
<th>HEAD_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Web and Information Technologies</td>
<td>Building A.5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Operational Research</td>
<td>Building A.3</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Artificial Intelligence</td>
<td>Building F.6</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 2.1: Table LABORATORIES, gathering the different laboratories

<table>
<thead>
<tr>
<th>RID</th>
<th>RNAME</th>
<th>LAB_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Doe</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Will</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Robert</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Smith</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Jack</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Tom</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>Andrea</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>Sophie</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>Alice</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2.2: Table RESEARCHERS, regrouping the different researchers

2.1 Openlink Virtuoso Universal Server

2.1.1 Introduction

OpenLink Virtuoso Universal Server is a cross-platform universal server. It acts as a virtual database engine managing a large number of databases types: DB2, SQL Server, Oracle, Sybase, etc. It is completely transparent to the user, who can easily access data distributed among a heterogenous set of server.

In addition to providing a single interface for databases, OpenLink Virtuoso Universal Server supports Web and data access standards: XML technologies (XPath expressions, XSLT, XML storage), Web services technologies (WSDL, UDDI, SOAP), WebDAV, SMTP, JDBC and ODBC, ... It runs on barely every common operating systems, namely Windows (95, 98, NT, 2000), Linux (Intel, Alpha, Mips, PPC), MacOS X, Solaris, FreeBSD, UnixWare, IRIX, ...

The roles and possibilities of OpenLink Virtuoso Universal Server are summarized on the diagram below (Fig. 2.1).

[http://docs.openlinksw.com/virtuoso/WhatIsVirtuoso.html]
Translation mechanisms In the context of this thesis, we are especially interested in the RDF data management part of Virtuoso. Virtuoso manages this by providing linked data views of relational data. They can be obtained by using Virtuoso’s declarative Meta Schema language, built over SPARQL, for mapping SQL data to RDF ontologies so that they can be queried by SPARQL. An example of such a mapping is provided in Listing 2.1.

```sparql
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ulb: <http://ulb.ac.be/db/> .
@prefix ulb_qs: <http://ulb.ac.be/db/quad-storage/> .

create iri class ulb:lab_iri "http://ulb.ac.be/db/laboratory/%d" (in LID integer not null) .
create iri class ulb:researcher_iri "http://ulb.ac.be/db/researcher/%d" (in RID integer not null) .

create quad storage ulb_qs:default
FROM UNIVERSITY.LABORATORIES
FROM UNIVERSITY.RESEARCHERS
{
  create ulb_qs:laboratories
  {
    ulb:laboratory_iri (LABORATORIES.LID)
    rdf:type ulb:Laboratory ;
    ulb:lname LABORATORIES.LNAME ;
    ulb:llocation LABORATORIES.LLOCATION ;
  }
}
```
ulb:directedBy ulb:researcher_iri(RESEACHERS.RID) ;
}
create ulb_qs:researchers
{
ulb:lab_iri(RESEARCHERS.RID)
rdf:type ulb:Researcher ;
foaf:name RESEARCHERS.RNAME ;
ulb:worksIn ulb:lab_iri(LABORATORIES.LID) ;
}

Listing 2.1: OpenLink Virtuoso example. foaf is the namespace Friend Of A Friend that describes persons.

A particularity of Virtuoso is that it does not store triples, but quads, which use a graph field indicating the belonging of a triple to an application or a resource. Quads are formed thanks to quad map patterns. Quad map patterns are the rules managing the translation process. They define the transformation of a column (or a set of columns) into triples. The quad map pattern is composed of four parts, specifying how to derive triple field from the SQL data [4].

In the mapping example, we use the so-called concise form of the Meta Schema language. We could have used the verbose one, but takes more places (the concise form is to the original one what turtle notation is to n3 in RDF).

A virtuoso mapping is divided in three main parts. First, the 'sparql' tag has to be given and can be followed by a series of prefixes that will be used later in the document. In our case, we use rdf (for the rdf:type predicate), foaf (for researchers names) and our own namespace ulb, addressing the description of all the other concepts.

Second, the different IRI classes have to be created. Our example requires to create two IRI classes, namely for laboratories and researchers. The IRI creation is made by replacing ad-hoc parts in a formatted string by values in columns. The concerned columns are specified in parenthesis alongside the string.

Third and final, a quad storage can be created. It will contain the different quads generated by quad patterns (two in our case). A quad map storage is very useful when dealing with a huge amount of quad map patterns, so they can be manipulated as sets. A quad storage is related to relational tables, which are the data sources for all the quad map patterns defined in this very quad storage (here, we are using UNIVERSITY.LABORATORIES and UNIVERSITY.RESEARCHERS). If needed, aliases may be used, like in SQL.

Virtuoso does not dump the relational data into RDF, but provides an RDF view that can be queried with SPARQL. In the following example, we query the RDF view to retrieve the subordinates of the head chief of the Artificial Intelligence laboratory.
The query answers evidently two names, Doe and Will, who are researchers but not directors.

### 2.1.2 Conclusion

Openlink Virtuoso is a huge universal server that centralizes a lot of functionalities that are usually distributed among multiple servers and it is designed to work on most of the existing architectures.

One of the module is built over SPARQL and allows users to create mappings. These mappings are a bridge, an interface, between SPARQL queries and relational data. These can be of any kind supported by Virtuoso (see Fig. 2.1).

### 2.2 D2R Server

#### 2.2.1 Introduction

D2R Server is a project developed at the Freie Universität Berlin. It has been initiated late 2006 (first results November 2006). In section 2.2.2, we will detail the architecture of the software and the different components underlying to this specific architecture. We will see that a current standard adopted by the W3 Consortium is based on a mapping language of D2R Server.

The goal of D2R server is publishing of relational databases content on the semantic web, namely in a RDF form. The base of functioning of D2R Server relies on a mapping mechanism to translate relational data (SQL, for instance) into resources and properties. Moreover, D2R server provides functionalities to Web agents to query RDF data with SPARQL and retrieve XHTML and RDF representation of those data (9).

#### 2.2.2 Architecture & functioning

The D2R distribution is divided in two main parts. The first one, which is the actual server part, is the interface between the D2RQ engine, which is the logical core of the application in charge of managing the databases (RDF and non-RDF) and the mapping files, and the outside world, connected to the Internet.

The D2R Server provides three different interfaces for data exchange, using the HTTP protocol: SPARQL for semantic queries, RDF for linked data browsers and HTML for ‘normal’ browsers.

The second part, and probably most important part, of the application is the D2RQ Engine. It manages the translation process from relational data to RDF. To do so, it uses a D2RQ Mapping file (the D2RQ Mapping process is explained in Sec. 2.2.4). Its purpose is to define a user-defined mapping allowing the translation which targets a chosen vocabulary.
2.2.3 D2RQ mapping example

```sparql
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#>.
@prefix map: <http://ulb.ac.be/db/d2r/example.ttl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix ulb: <http://ulb.ac.be/db/>.
@prefix : <http://ulb.ac.be/db/>.

map:UnivDB a d2rq:Database;
  d2rq:jdbcDSN "jdbc:mysql://ulb.ac.be/db/University";
  d2rq:jdbcDriver "com.mysql.jdbc.Driver";
  d2rq:username "adminsyst";
  d2rq:password "S3cr3tP4ssw0rD!";

map:Researcher a d2rq:ClassMap;
  d2rq:dataStorage map:UnivDB;
  d2rq:class ulb:Researcher;
  d2rq:uriPattern "http://ulb.ac.be/db/reseacher/@@RESEARCHERS.RID@@".

map:researcherName a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:Researcher;
  d2rq:property foaf:name;
  d2rq:column "RESEARCHERS.RNAME";
  d2rq:datatype xsd:string.

map:researcherLab a d2rq:PropertyBridge;
```

Figure 2.2: The D2R Server base architecture
Image source: [http://d2rq.org/images/architecture.png](http://d2rq.org/images/architecture.png)
In this example, we first specify the different useful prefixes that will be used. The map is the namespace of the mapping file and it will not appear in the resulting mapped data; rdfs, rdf, foaf and xsd are namespaces defined by other projects or organizations; ulb is our own namespace. It defines the concepts characterizing the ULB (researchers, laboratories, and so on). The blank prefix is equivalent to the ulb one.

In the second part (line 18), we declare a Database object, with the location, the driver that should be used and a username with a password as well.

Finally, we specify the different mappings. There are ClassMaps (lines 25, 42) which are rules to create RDF subjects, namely the researchers and laboratories from their respective id, and there are PropertyBridges (lines 30, 36, 47) whose purpose is the creation of RDF predicate-object couples. We can emphasize that each PropertyBridge belongs to one ClassMap (d2rq:belongsToClassMap) and that each ClassMap is related to one data source (d2rq:dataStorage), in this case map:UnivDB.

To keep the links between data offered by foreign keys, references to resources created by other ClassMaps can be made by specifying which one is referenced (lines 39, 50) and what constraints are applied (d2rq:join, lines 40, 51).

The result of this mapping can be found in appendix B.

2.2.4 D2RQ Mapping Language

D2R Server uses the language D2RQ Mapping Language, addressing the specific needs of the application. The role of a D2RQ Mapping is to specify how RDF triples have to be created from a relational data source. The goal of the following description is not to explain all the possible properties for each class composing the mapping language, but rather outline the most relevant of them and explain their main purpose.
The main element of the vocabulary, which constitute the core of a D2RQ mapping, is the \texttt{d2rq:ClassMap}. It represents the set of relational elements that will be translated to a certain class of resources. It also defines how the instances of those class are identified. A \texttt{d2rq:classMap} is linked to a \texttt{d2rq:Database} element.

The D2R Mapping Language provides four mechanisms for identifying instances of classes.

**URI patterns and relative URI patterns.** A URI pattern is a generic string which is instantiated with values contained in the database. For example, the pattern
\begin{verbatim}
http://myexample.org/researcher/@@RESEARCHERS.RID@@
\end{verbatim}
will replace the part between '00' by values in the \texttt{RID} column of the \texttt{RESEARCHERS} table. For relative URI patterns, the process of URI creation remains identical to the previous one, unless that a relative URI is given and it must be combined with a base URI provided by the processing environment.

**URI columns.** In some cases, database columns already contain URIs (for example, a column containing webpage links or email addresses). Values can therefore be used directly as resources in the RDF graph.

**Blank Nodes.** Those existential qualifiers can be used to describe resources that cannot be named and that has certain properties. They are created from one or more columns and a distinct blank node is generated for each distinct set of values.

**URI expressions.** One can generate URIs with a valid SQL expression. The only constraint is that the result of this expression must produce a valid URI.

Each \texttt{classMap} possesses a collection of \texttt{d2rq:propertyBridges}. They are giving properties to bind properties to resources generated by class maps (to resume, \texttt{d2rq:propertyBridge} gives the predicate and object parts of a triple). Even though values of properties are just literal in most of cases, blank nodes or resources are also accepted.

We shall note that if a column present a \texttt{NULL} value for a row, no property is created from this value.

\texttt{d2rq:propertyBridge}s present the same URI generation mechanisms that those used in class maps. Join between tables can performed to get necessary information present in other tables (this appens quite often if database tables are normalized).

For literals, language or datatype can be precised as well.

Another possibility offered by D2R Mapping Language are the conditionnal mappings. They intend to solve the issue induced as some information contained in the database should not be accessed, some information are outdated or the generation of resources relies on the value in the tuple. The functioning of the \texttt{d2rq:condition} is exactly the same as a \texttt{WHERE} clause.

### 2.2.5 Summary

The D2R Server Tool is in development since 6 years. The current version (May, 2012) is D2R Server v0.8. It provides a lot of functionalities such as RDF querying, RDF browsing, RDF
dumping from relational data and automatical mapping generation, which help the developer to start his mapping.

Regarding the mapping language, D2RQ Mapping Language, a positive point is that it fits in the RDF standard; each mapping will be a valid RDF graph. However, one can deplore the fact that the vocabulary is composed of a very large amount of words. The learning and the implementation of such a language gain hence in complexity.

Finally, D2R Server does not support officially database systems other than Oracle, MySQL, PostgreSQL, SQL Server, HSQLDB and Interbase/Firebird. The adding of XML sources would thus not work.

2.3 Direct Mapping

2.3.1 Introduction

There are naturally a plenty of methods to translate relational data. The first one, presented here, is the Direct Mapping, under standardization by the W3C, which is probably the simplest one in essence.

A direct mapping takes in input a relational schema and an instance of this relational schema (more simply, a relational database) and gives as output a RDF graph. The result can then be queried with SPARQL or any other RDF query language.
At this time, Direct Mapping has passed the stage of Working Draft of the W3C and is a candidate recommendation, which means that conceptual developers are thinking that the version has evolved and is stable enough to encourage developers to implement the standard.

In this section we will examine in detail the possibilities offered by Direct Mapping. Those will be illustrated with the running example from the previous section (Sec. 2), namely the database of the University, with a table describing the researchers and another the laboratories.

2.3.2 Functioning

2.3.2.1 Simple Case

The translation process of Direct Mapping is simple and straightforward. For each tuple of each table in the database, the same routine will be applied, the final goal being the obtaining of a set of triples \((s, p, o)\).

The subject part \((s)\) is gained by using the primary key of the tuple. It is aggregated with a base URI, the table name and the primary key column name to form a valid URI.

The predicates are acquired by a similar mechanism; the column names are concatenated with the table name and the base URI.

Objects are directly considered as literals.

To resume this simple case, if the tuple contains \(n\) elements, \(n + 1\) triples will be generated, the \(+1\) being the subject, the predicate \(\text{rdf:type}\) and the object formed by the concatenation of the table name and the base URI. The \(n\) tuples may be represented by this formula:

\[
(s_i, p_j, o_{i,j}) = \langle \text{tuple}_i(PK) \rangle < \text{schema}(j) > "\text{tuple}_i(j)". \tag{2.1}
\]

where \(i \in [1, \#\text{tuples}]\) and \(j \in [1, \text{size}(\text{schema})]\).

The result of translating the table shown on the figure 2.4 is the following:

```
1 @ base <http://cofromen.example.org/DB/> .
2 @ prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

3 <Tablename/ID-1> rdf:type <Tablename> .
4 <Tablename/ID-1> <Tablename:ID> 1 .
5 <Tablename/ID-1> <Tablename:P1> "Value 1" .
6 <Tablename/ID-1> <Tablename:P2> "Value 2" .
7 <Tablename/ID-1> <Tablename:P3> "Value 3" .
8 <Tablename/ID-1> <Tablename:P4> "Value 4" .
9 <Tablename/ID-1> <Tablename:P5> "Value 5" .
10 <Tablename/ID-2> rdf:type <Tablename> .
11 <Tablename/ID-2> <Tablename:ID> 2 .
12 <Tablename/ID-2> <Tablename:P1> "Value 2" .
13 <Tablename/ID-2> <Tablename:P2> "Value 6" .
14 <Tablename/ID-2> <Tablename:P3> "Value 9" .
15 <Tablename/ID-2> <Tablename:P4> "Value 9" .
```


2.3.2.2 Link with other tables, referencing candidate keys

In practice, translating a database does not resume itself to translate a single table, because data is normalized or simply too complex to fit in one table.
While translating, if a foreign key is encountered, it will generate a triple with a predicated formed by the concatenation of the table name and the referencing column names. The generated object will be a resource formed by the aggregation of the referenced table name, its tuple identifier column name and its tuple identifier value. The running example would give the following partial result:

```
1@ base <http://ulb.ac.be/db/> .
2@ prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3
4   @ base <http://ulb.ac.be/db/> .
5@ prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
6
7   @ base <http://ulb.ac.be/db/> .
8@ prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
9
10@ base <http://ulb.ac.be/db/> .
11@ prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
12
13@ base <http://ulb.ac.be/db/> .
14@ prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
15
16@ base <http://ulb.ac.be/db/> .
17@ prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
18
```

**Listing 2.4:** Partial result of the direct mapping applied on the database UNIVERSITY, demonstrating the generation of triples with references between tables. Turtle notation.

It may happen that the foreign key is composed of several columns and references a unique key which is not the primary key of the referenced table (that is, a candidate key). In that case, the process slightly changes; each value of the column constituting the key generates a RDF literal with its corresponding predicate (formed with the column names) and another triple is created, with a predicate gathering all the column names (separated by a dot character) and an object being a resource formed with the tuple identifier from the referenced table.

To illustrate this mechanism, we will look at the example given in the W3C candidate recommendation [15]:

```
1@ base <http://foo.example/DB/> .
2@ prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
3
4@ base <http://foo.example/DB/> .
5@ prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
6
7@ base <http://foo.example/DB/> .
8@ prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
9
10@ base <http://foo.example/DB/> .
11@ prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
12
13@ base <http://foo.example/DB/> .
14@ prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
15
16@ base <http://foo.example/DB/> .
17@ prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
18
```

**Figure 2.4:** This picture represents the simplest case of direct mapping, without any reference to other table.
Table 2.3: Tables People, Department and Addresses, connected with some fancy foreign keys

<table>
<thead>
<tr>
<th>PEOPLE</th>
<th>ref ADDRESSES</th>
<th>ref DEPARTEMENT NAME,CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
<td>ref</td>
<td>ref</td>
</tr>
<tr>
<td>ID</td>
<td>PEOPLE(ID)</td>
<td>DEPARTEMENT(ID)</td>
</tr>
<tr>
<td>7</td>
<td>Bob</td>
<td>accounting Cambridge</td>
</tr>
<tr>
<td>8</td>
<td>Sue</td>
<td>NULL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>ref PEOPLE</th>
<th>ref ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
<td>UNIQUE KEY</td>
<td>ref</td>
</tr>
<tr>
<td>ID</td>
<td>NAME</td>
<td>CITY</td>
</tr>
<tr>
<td>23</td>
<td>accounting</td>
<td>cambridge</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDRESSES</th>
<th>ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>CITY</td>
</tr>
<tr>
<td>18</td>
<td>Cambridge</td>
</tr>
</tbody>
</table>

Listing 2.5: Direct Mapping on tables with foreign keys referencing candidate keys. Example from W3C recommendation [15]. Notation 3.

As last remark about the functioning of Direct Mapping, one can note that when a NULL value occurs in a table, no object is generated and the translation process passes this singularity. This might seem trivial but this particularity must absolutely kept in mind while implementing this standard.

2.3.3 Conclusion

Direct Mapping is an easy-to-learn and easy-to-use standard, designed for an easy implementation. Its main advantage is that there is no need for extra information from a user to translate his database once he possesses the implementation in his hands. Most of the other translation mechanisms will require a mapping to achieve the process.
Furthermore, Direct Mapping produces in output a RDF graph that reflects the exact same structure as that of the database. The final output contains thus every references and links that were present in the relational data, without any losses.

However, one major drawback of Direct Mapping, which derives from its first quality is that there is no way to use existing vocabularies to describe the data, since all the predicates are constituted of the column names. We are losing de facto a big advantage allowed by RDF, the possibility to infer knowledge with ontologies. Despite this major inconvenient, Direct Mapping is still very useful to validate some complexer translation models, such as R2RML or any other.

2.4 R2RML

2.4.1 Introduction

R2RML, or Relational To RDF Mapping Language, is a language defined by the W3 Consortium, which is “designed to express customized mappings from relational databases to RDF datasets”.

Those mappings allows the author to view existing relational data in a RDF data model, in the structure and with the vocabulary he finds appropriate. As we have seen, in the Direct Mapping (Sec. 2.3), such an operation is not permitted by the translation mechanism ; R2RML offers thus more personalization possibilities.

At this time, R2RML is a candidate recommendation of the RDB2RDF Working Group of the W3C. The first public version of the language has been released in 2010, October. That version has experienced three updates ; the first one in 2011, March, the second one, which was the last working draft in September, 2011 and the final one that is the current one in 2012, February.

In this section will be detailed the syntax and the vocabulary provided by R2RML, illustrated with the running example.

2.4.2 Technical details

A relational to RDF mapping is written in a mapping file containing a “text” that is semantically and syntactically correct according to the R2R Mapping Language specifications. This text will be written in turtle notation. The mapping file itself is a RDF graph, and can be manipulated in this way in a later use (for example while it is parsed) ; moreover, the RDF formalism is not only used as target data model but also to represent the mapping.

2.4.2.1 Global example of R2RML mapping

```turtle
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ulb: <http://ulb.ac.be/db/>.

<#researchers_TM>
  rdf:type rr:TriplesMap;
```

[http://www.w3.org/TR/2011/WD-r2rml-20110920/](http://www.w3.org/TR/2011/WD-r2rml-20110920/)
rr:logicalTable[
  rr:tableName "RESEARCHERS";
];

rr:subjectMap[
  rr:template "http://ulb.ac.be/db/researcher/{RID}";
  rr:class ulb:Researcher;
];

rr:predicateObjectMap[
  rr:predicateMap[
    rr:constant ulb:rid;
    rr:termType rr:IRI;
  ];
  rr:objectMap[
    rr:column "RID";
    rr:datatype xsd:integer;
    rr:termType rr:Literal;
  ];
  ];

rr:predicateObjectMap[
  rr:predicate foaf:name;
  rr:objectMap[
    rr:column "RNAME";
    rr:datatype xsd:string;
  ];
];

rr:predicateObjectMap[
  rr:predicate ulb:worksIn;
  rr:refObjectMap[
    rr:parentTriplesMap <#laboratories_TM>;
    rr:joinCondition[
      rr:parent "LID";
      rr:child "LAB_ID";
    ];
  ];
].

<#laboratories_TM>
a rr:TriplesMap;

rr:tableName "LABORATORIES";

rr:subjectMap[
  rr:class ulb:Laboratory;
  rr:template "http://ulb.ac.be/db/laboratory/{LID}";
];

rr:predicateObjectMap[
  rr:predicate ulb:lid;
  rr:objectMap[
    rr:column "LID";
  ];
];

rr:predicateObjectMap[
  rr:predicate ulb:lname;
  rr:objectMap[
    rr:column "LNAME";
    rr:language "en-GB";
  ];}
Listing 2.6: R2RML global mapping example

2.4.2.2 General overview

Concretely, a R2R mapping is a list of one or more rules that specify how the relational data has to be translated. Those rules are named *triples maps*. In essence, a mapping file will look like this:

1 @prefix rr: <http://www.w3.org/ns/r2rml#>.
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
3 ## Other prefixes
4 <#TriplesMapName1>
5  a rr:TriplesMap;
6  ## Other content
7  .
8 <#AnotherTriplesMap>
9  a rr:TriplesMap;
10  ## Other content
11  .

Listing 2.7: Mapping syntax, example 1

In each of these triples maps, we will thus find the body of the rule. The rule is divided in three main pieces.

First, one needs to precise the data on which the translation is applied. This relational data takes the form of a table, known as the *logical table*. Later, we will see how such a logical table may be defined.

Second, each RDF triple needs a subject. Therefore, the second specification of the triples map defines how to create the subject for each tuple of the logical table. This is called the *subject map*.

Finally, to achieve the mapping, we need to add a rule that generates the predicate part and the object part of the triples. Again, it is naturally named the *predicate-object map*.

At this point, it is important to precise that for each triples map, there will be one and only one logical table, one and only one subject map and zero or more predicate-object maps, as many as needed. Indeed, it would not be logical to take two sources of data in the same rule (we would likely have problem to generate the subject) or precise more than one way to generate a subject of a triple. However, it is natural to have the possibility to set as many predicate-object maps as needed, following the information we want to be present in the final RDF graph.
The base skeleton of the mapping file being known, in the following we will detail the main possibilities of each part of the rule. For reasons of clarity, we shall follow the order in which the different rules appear in the mapping file, namely the logical table, the subject map and finally the predicate-object map.

To make the understanding of the syntax easier, we will use the running example.

### 2.4.2.3 Logical tables

As we have seen, the first part of the mapping rule is the definition of the base relational data. Instead of working directly on a database entity, such as a relation, R2RML provides an abstract interface to relational data through the logical table. This table includes all the information we want to be translated with the current triples map and it can defined in several ways.

R2RML gives a precise definition of a logical table and its different components:

- “A logical table is a possibly virtual database table that is to be mapped to RDF triples. A logical table is either a SQL base table or view or a R2RML view.”[^3]
- “A logical table row is a row in a logical table”. Everytime that we will use the term tuple in the context of mapping relational data, it has to be considered as a logical table row.[^4]
- “A column name is a the name of a column of a logical table”. This column name must be a SQL identifier (which means that it must match the `<identifier>` production in the SQL2 language definition).[^4]

Before going further in the different types of logical tables, a schema representing the different properties of a logical table could help

In the most simple case, a logical table is a SQL base table or a SQL view. This kind of logical table shall contain relational data as it is in the input database. For instance, a database with a table or a view named `student` will be ‘transformed’ in logical table like this, grabbing all the values of the table/view without any selection:

[^3]: http://www.w3.org/TR/r2rml/#logical-tables
[^4]: http://www.w3.org/TR/r2rml/images/logical-table.png
Figure 2.5: The different properties that a logical table can have. The prefix `rr` is the same than in every examples of mapping.

Listing 2.9: Logical table, `rr:tableName`

There are two restrictions with those base tables and sql views:

1. there will be only one `rr:tableName` property, no less no more
2. the value of `rr:tableName` must be a valid schema-qualified name (sequence of one to three valid SQL identifiers separated by a dot '.') that references an element of a table or a SQL view in the input database.

The second type of logical table are the R2RML views, who are SQL queries executed on the input database. The effective query is given by a single property `rr:sqlQuery`, whose value must be a valid SQL query. In addition, the SQL version may optionally be specified, through a valid IRI (for instance `http://www.w3.org/ns/r2rml#2008` represents the Core SQL 2008).

The difference between a classic SQL view and a R2RML view is that a R2RML view can not be used in later SQL manipulations (see later in referencing object maps).

Here below is shown an example of a logical table constituted of three attributes (`eid`, `name`, `year`) of the table `student`.

Listing 2.10: Logical table, `rr:sqlQuery` & `rr:sqlVersion`
2.4.2.4 Term maps

Now that data is available in the logical table, the real work of translation begins here with the generation of triples terms, thanks to term maps. Subject maps, predicate maps, object maps and graph maps are all term maps, that is they have the function to generate parts of the future triples. They have a lot of properties in common.

As shown in Fig. 2.6, there are four kind of term maps:

- subject maps, that generate the subject part of the triple;
- predicate maps, that create the predicate part;
- object maps, that build the object term;
- graph maps

As we may see on the same Fig. 2.6, there are seven possible properties for any term map. In practice, one term map cannot present those seven; the three properties in blue on the figure are mutually exclusive: a term map is either a column-valued term map, a template-valued term map or a constant-valued term map. It is important to precise that a term map necessarily must have one and only one of the three properties.

Let us look a bit further to the three types of term maps:

**constant-valued term maps.** They are the simplest kind of translating rules. The term that will be generated is a constant, and therefore does not rely on a particular value in a tuple. Predicates are often constant-valued term maps (e.g., `rdf:type` or `foaf:name`).
column-valued term maps. They will generate terms with the value in a tuple at the corresponding column. However, there is a restriction by using this kind of rule: both subject map and predicate map must generate IRIs and not a literal. They can be column-valued term maps, but the column must contains only values that are already IRIs.

template-valued term maps. They are the last type of term maps. They allow to create IRIs (or any other result if needed) from values of any kind. The principle is quite simple: the mapping gives a special base string composed of a base string (that can be an IRI but not necessarily) mixed with some names between curly braces. The names between the braces must strictly match the actual column names of the logical table. The final value is generated by replacing the braces parts by the values of the tuple for the different attributes.

On the following listings, we are giving examples for the three types of term maps, that could be part of a triples map meant to translate the ‘RESEARCHERS’ relation from the running example database:

Listing 2.11: Constant-valued term map. The generated term is the always the same, in this case foaf:name

Listing 2.12: Column-valued term map. The column properties is associated with a column-name. The resulting values of the object map are the values in the specified column.
In addition, term maps may present the following properties:

**rr:termType.** This property specifies the type of term being generated, that is an IRI, a blank node or a literal. The associated objects are classes belonging to R2RML, namely **rr:IRI**, **rr:BlankNode** and **rr:Literal**. It is allowed to omit this property; in that particular case, the generated term will be considered as a literal if and only if (i) it is a column-based term-map, or (ii) a language property is specified, or (iii) a datatype property is given. Otherwise, the term will be an IRI.

**rr:language.** Since RDF gives the possibility to annotate a literal with a ISO language tag (e.g., "@en-US" or "@fr-BE", R2RML proposes a property to generate such literals. The property associated object is a literal with the language code (see line 66 in Listing 2.6).

**rr:datatype.** If no language tag is specified, the generated literal can be datatyped once. This is done by giving an IRI describing the wished datatype. Usually, one will use those provided by the XSD namespace.

**rr:inverseExpression.** This purpose of this last property, that can be associated with a column-based or template-base term map, is to retrieve original relational data from one of a triple term. An inverse expression is a string template object with unescaped curly braces containing some logical column names.

### 2.4.2.4.1 Subject maps

Subject maps are rules to generate subjects of triples. Therefore, they are a subtype of term maps: all the aforementioned properties can be applied. However, subject maps present several particularities:

- The **rr:termType** must be an IRI or a blank, literal being forbidden.
- Various IRI classes (**rr:class** properties) may be specified. Their purpose is to define the resource belonging to a particular class. They are associated with a resource as object. For instance, a researcher can present both classes **ulb:Researcher** and **foaf:Person**, specifying that he is employee of the ULB and, happily, a human being as well.

### 2.4.2.4.2 Predicate-Object Maps

Predicate-object maps are the another subtype of term maps. Their purpose is the creation of predicate-object couples in the triples. One can find zero or more in any triples map. Each **rr:_predicateObjectMap** contains a predicate map and an object map, respectively in charge of creating a predicate and an object. If those maps are constant-valued term maps, a shorter notation may be used (**rr: predicate** and **rr: object**). All the aforementioned constraints characterizing term maps are applicable to predicate-object maps.

### 2.4.2.4.3 Referencing other tables
Most of the time, database tables are linked with each other by using foreign keys. Therefore, R2RML provides a simple mechanism to link resources generated from different tables.

As a reminder, a triples map is defined for each type of resource that will be created. Each triples map possesses one logical table which might be an aggregation of several relational tables, thanks to R2RML views (SQL query). To create links between different triples map (i.e., create links between resources), R2RML provides a special type of object map: the referencing object map.

Referencing object maps ($rr$:refObjectMap) a way to generate an object which is a resource created by another triples map. They are blank nodes composed of the following elements:

- a $rr$:parentTriplesMap, that references an existing triples map in the same mapping file that generates the wanted resource.
- zero or more join conditions ($rr$:joinCondition). A join condition contains exactly two elements, namely $rr$:child and $rr$:parent. The former is associated with a column name of the logical table of the triples map containing the referencing object map. The latter is coupled with a logical column name of the referenced triples map.

The application of the join condition gives a set of relational tuples that is used by the subject map of the referenced triples map. From a SQL point of view, it is obtained by applying the following query:

```sql
SELECT * 
FROM (child-query) AS child, 
      (parent-query) AS parent 
WHERE child.{child-column1}=parent.{parent-column1} 
AND child.{child-column2}=parent.{parent-column2} 
AND ...
```

Listing 2.14: SQL query to obtain the set of tuples resulting of a $rr$:joinCondition
Listing source: [http://www.w3.org/TR/r2rml/#foreign-key](http://www.w3.org/TR/r2rml/#foreign-key)

On the previous listing, the child-query and the parent-query are the effective SQL query of the referencing and the referenced logical tables triples map. The child-column$\text{i}$ and parent-column$\text{i}$ are the column names in their respective join conditions.

```json
rr:predicateObjectMap [ 
  rr:predicate ulb:worksIn; 
  rr:refObjectMap [ 
    rr:parentTriplesMap #laboratories_TM; 
    rr:joinCondition [ 
      rr:parent "LID"; 
      rr:child "LAB_ID"; 
    ]; 
  ]; 
].
```

Listing 2.15: Join Condition example

On the previous example we create a joinCondition between the LID column in the LABORATORIES table and the LAB_ID column of the RESEARCHERS table.
2.5 db2triples

db2triples is an free open-source Java implementation of the Direct Mapping mechanism and the R2R Mapping Language (R2RML). This component has been developed and released by Antidot[^6] an enterprise developing tools for linked data, among others.

db2triples reaches the goals established by the w3c[^7], namely integrate data from SQL databases into the linked data, give the possibility to manipulate them with SPARQL and gather them with other data sets. This implementation is maintained by the community on GIT repository[^8].

The use of db2triples is quite easy. It offers a command-line interface. Two entry-points for the application are provided: one for using Direct Mapping and the other to use R2RML. Both translation processes require the specification of a database in input (with eventual parameters for the connection). When using R2RML, a mapping file must be specified as well. db2triples dumps the result of the translation into user’s file.

db2triples implements an first and outdated version of R2RML (2011, March). At the current time (2012, May), no updates have been made[^9].

2.6 Triplify

2.6.1 Overview

In order to publish Linked Data from relational sources, a light-weight application called Triplify has been created. The main purpose of triplify is to promote Semantic Web usage by ease the publishing of RDF triples from existing Web applications. It aims also to manage large amount of data with a flexible and scalable approach. In a lesser degree, Triplify provides pre-existing mapping for popular Web applications (Drupal, WordPress).

[^Auer et al.]: identify three major obstacles to Linked Data from relational databases publication:

- Ambiguity between private and public data: Web application with relational database backend contains information that should not be published on Web (email, passwords) or that could not be published (technical parameters). The distinction between those data and revelent data is sadly mandatory and very hard (even impossible ?) to apply.

- Missing database schema description: expansion of badly designed databases schemas and lack of clear constraints definitions are making the syntactic identification of those constraints difficult to succeed.

- Reuse of existing vocabularies: tables and columns names might not be mapped correctly to vocabularies due to insufficient semantics in the database schema.

[^http://www.antidot.net/]:
[^http://antidot.net/fr/Actualites/Produit/Antidot-fournit-db2triples-en-Open-Source]:
[^https://github.com/antidot/db2triples]:
[^https://github.com/antidot/db2triples/blob/master/README]:
These obstacles appear to be the main barrier to publish relational content as RDF. The main objective of Triplify is to lower this barrier by proposing tailored RDB-to-RDF mapping solution. Unlike other RDF publishing tools like D2R Server or OpenLink Virtuoso Server, Triplify does not use or define a special mapping language but uses SQL to transform query results into Linked Data.

From a technical point of view, the use of SQL as a mapping language is justified by the following reasons:

- Maturity of the language: SQL has proven its qualities as relational structure shifter and is the result of years of upgrades and improvements.

- Role of the database: the expensive operations (like complex joins) are left to the database. Databases have mechanisms to handle and optimize complex operations (indexing, query optimization, ...), which make Triplify gain in scalability.

- Knowledge of SQL: most of database systems administrators are familiar with SQL.

### 2.6.2 Translation mechanisms

The first step of the translation process is a relational transformation of the data with the SQL language, to create an appropriate view for generating linked data. An appropriate view is illustrated in Fig. 2.7 and defined as following:

- the first column must contain a value that unequivocally defines a record, that is an identifier (the primary key, for example). The value will be used to generate the URI of triples subjects.

- the column names will be used to generate predicates.

- cell values will generate values constituting the objects of triples.

![Figure 2.7](image)

Figure 2.7: Triplify translation mechanism. The subject is the first column; each column is mapped to a predicate; each value of the tuple is mapped to an object.
One might think that the process described above is similar to Direct Mapping (see Sec. 2.3, p.26). However, the difference between these two mechanisms lies in the SQL mapping used in Triplify. Once this relational shift applied, the generation of triples is similar (see Fig. 2.4, p.28).

In its configuration, Triplify allows the use of existing vocabularies like foaf or rdfs. They must be declared first in the configuration containing all the necessary namespaces. Then, they can be used in column names (and the base URI is therefore not applied to column name to generate a valid URI). For instance, we could map the name of researchers to foaf:name (Friend Of A Friend namespace) like this:

```
SELECT R.RID, R.RNAME AS 'foaf:name', R.LAB_ID FROM RESEARCHERS R
```

This would give us a schema similar to `[RID, foaf:name, LAB_ID]`

By default, the generation of objects produces literals. Since sometimes one expects resources, Triplify setted up a system of reference to other views by using a `->'set_name'` notation, where `set_name` references another set of queries represented by `name` (in this case, this is PHP notation, since Triplify is implemented in PHP). The rule used for generating URIs in the other set is applied to the value in the current set. Here is an example with the running example, written in PHP.

```php
$triplify['queries'] = array(
    'researchers'=> array(
        SELECT R.RID, R.RNAME AS 'foaf:name', R.LAB_ID AS 'worksIn->laboratories'
        FROM RESEARCHERS R
    ),
    'laboratories'=> array(
        SELECT L.LID, L.LNAME, L.LLOCATION, L.HEAD_ID AS 'directedBy->researchers'
        FROM LABORATORIES L
    )
);
```

**Listing 2.16:** PHP example of triplify mapping, with references to other sets of queries.

In this example, we have two sets of queries (researchers and laboratories). Each contains only one query but more would have been acceptable. The references to the other sets are namely `worksIn->laboratories` and `directedBy->researchers`.

Finally, datatypes can be directly retrieved from the metadata of the database (a column with integer will give a literal with the `^^xsd:integer` suffix) and used with the XSD namespace. Languages tags can be precised in the SQL mapping. For instance, one could describe the name of a laboratory in English by doing

```
SELECT L.LNAME AS 'rdfs:label@en-GB' FROM LABORATORIES L
```

### 2.6.3 Summary

Triplify appears to be a good system for translating relational data into linked data. It presents a lot of serious advantages, such as the maintain of expansive computation to the database, the use of the mature SQL language and its high scalability. However, there are several drawbacks to use Triplify.
First, the standard is maintained by very few people and documentation is hence really poor, unlikely standards developed by big groups such as W3C.

Second, the intention of developers is to not implement extensions such as SPARQL support or other improvements.

Finally, even if is an express wish of the developers, it is sad that a mapping for RDF is not an RDF graph itself. This would have allowed some extra possibilities of data processing and reuse with a mapping suited for Web 3.0.

2.7 Comparative study

In this chapter, we have discussed the (non-exhaustive) different standards and implementations that define methods to translate relational data into RDF. Each of those have their advantages and drawbacks. Therefore, we propose a comparative table (see Tab. 2.4) retaining the aforementioned existing solutions of the state of the art, that summarizes their main characteristics. From this table, we will justify the choices made to solve the use cases.

2.8 Conclusions

In this chapter, we have developed the State of the Art concerning the translation of relational data into RDF. We approached various software implementations such as OpenLink Virtuso Universal Server, D2R Server, db2triples and Triplify and some theoretical standards developed by the W3C as well, namely Direct Mapping and R2RML. We concluded this State of the Art by comparing those different approaches.

In the following section, we will develop different use cases. We will then present the ontology dealing with cultural events that was created for the OSCB project. We will finally conclude with the requirements deductible from the use cases, the comparative study and the ontology.
<table>
<thead>
<tr>
<th>Component</th>
<th>Advantages</th>
<th>Drawbacks</th>
</tr>
</thead>
</table>
| OpenLink Virtuoso Universal Server | - All-in-one server  
- Supports plenty of connection types  
- Integration with other services  
- RDF Views (VS. RDF Dumps) | - Heavy-weight  
- Commercial solution  
- Unmodifiable implementation  
- No support for XML sources |
| D2R Server                 | - RDF Views and RDF Dumps  
- Customizable mappings  
- Complete mapping language | - D2RQ Mapping language hard to implement  
- No support for XML sources  
- Connection to datasources implemented in the mapping language |
| Triplify                   | - Fast execution (SQL level)  
- Easy to learn and use  
- Customizable “mapping” | - Impossible to use with XML  
- Mapping language in SQL (not RDF-graph) |
| db2triples                 | - Light-weight  
- Free and open-source  
- Direct mapping and R2RML implementation | - R2RML version outdated  
- No support for XML sources |
| Direct Mapping             | - Easy to learn  
- Easy to implement  
- W3C recommendation | - No customizable mappings  
- No support for XML sources |
| R2RML                      | - W3C recommendation  
- Customizable mapping  
- Concise mapping vocabulary | - No support for XML sources |

Table 2.4: Comparative table summarizing the main characteristics of the existing state of the art components.
Chapter 3

Use Cases and requirements

In this chapter we will dive further in the description of the use cases by giving explanations about the different relational data structures and the interest to translate those data. Then we will explore the ontology describing the cultural concepts common to the various actors. Finally, we will deduce the requirements of our implementation from the comparative study of the state of the art, the ontology and the use cases.

Contents

<table>
<thead>
<tr>
<th>3.1 Use cases description</th>
<th>44</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2 Cultural and location ontology description</td>
<td>46</td>
</tr>
<tr>
<td>3.3 Requirements</td>
<td>47</td>
</tr>
</tbody>
</table>

3.1 Use cases description

3.1.1 A use case: BOZAR

The BOZAR database is composed of seventy tables (see Appendix A) and weigh approximately 130MB. Among those tables stand some particular tables ending with the _lng suffix. Each of these tables has the exact same schema: [id, lng, field, content]. Tuples of these tables can be seen as triples, where the id column contains the subject, the field column the predicates and the content column the object. For convenience, we will henceforth name these special tables semantized tables.

The semantized tables contain the actual information about special classes of objects (activities, personalities, opuses, and so on). For each semantized table, we find a twin sister table (without the _lng suffix) that contains keys pointing to other tables and information with barely no semantics.

The BOZAR database contains tables storing data about the layout of the website as well. This information should of course not be translated into RDF since it is not a reusable information. The various tables that should be translated are the following are the tables preceded by the prefix ‘**’ specified in appendix A.

The translation of the BOZAR database represents a great challenge for several reasons. First, the number of tables is significant. Second, the architecture is complex, since data about
same concepts is distributed over a lot of tables (e.g., activities). Third, some tables have an unconventional schema that can lead to difficulties while translating (ling tables). Fourth and final, the semantics of the metadata is not reflected by the attribute names of the schemas.

3.1.2 A use case: Agenda.be

The agenda.be data concerning events and institutions are published in XML. They are both simple XML trees. For the sake of simplicity, we will detail the tree of the events, but the architecture remains identical for institutions. The root node is Events; from a relational point of view, the root node is the table. The child nodes of the root are element nodes Event_Instance; they are the tuples of the table. Each of these tuples contains 287 descendants elements nodes. Each of those which possesses a text node as child corresponds to a column in the relational model. Their text nodes are hence the corresponding values for the tuple.

For instance, if we compare the XML tree to a relational table, a tuple 'begins' at line 4. Its first value is 233999 for the EventID attribute. The second one is Sarah. for the Event_Title_FR attribute, etc.
The XML files we used to translate weigh approximately 6MB and are composed of ~450 tuples (Event_Instance nodes).

One objective of the thesis is to translate both events and institutions XML data into linked data in respect with the ontology describing the cultural things. If this objective is reached, agenda.be would be able to publish its data and directly reuse the linked data of other cultural actors without having to translate them into XML.

### 3.2 Cultural and location ontology description

The match the objectives of the OSCB project, an ontology describing the cultural domain is necessary. Moreover, the different actors involved in the project had to agree about the various concepts defined by this ontology. Indeed, the definition of an event (and the other concepts linked to it) changes from an partner to another. The creation of the ontology has been done by other participants in the OSCB (VUB STAR lab), but we participated in the maintenance of this one, since during the mapping process some concepts appeared to be missing or incomplete.

Actually, this issue led us to create an extension to the cultural domain ontology. On one side, in the use cases data, there are a lot of references to places and locations of event. On the other side, the rise of new technologies in image recognition and semantic web, applications allowing to find every cultural events organized nearby a location taken in picture, for example, are likely to emerge. Therefore, there is a need to enrich the cultural ontology with one describing locations. The resulting addendum to the cultural ontology is provided in the next chapter (Sec. 4.3.2).

The cultural ontology intends to define concepts and link them by establishing a hierarchy and relations. In the following code snippet, we give a insight of what it looks like:

```xml
<owl:Class rdf:about="/gospl/ontology/2#Event">
  <rdfs:label>Event</rdfs:label>
</owl:Class>

<owl:ObjectProperty rdf:about="/gospl/ontology/2#Event_has_Title">
  <rdfs:label>has Title</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="/gospl/ontology/2#Event_is_related_to_Discipline">
  <rdfs:label>is related to Discipline</rdfs:label>
</owl:ObjectProperty>
```

46
In this example, the ontology is defined by using OWL (Web Ontology Language), a language using RDF whose purpose is to define concepts and their associated properties. The first block defines the class of objects Event, associates a URI to it (gospl:Event) and specify how to represent it in a RDF graph (with a label “Event”). The two last blocks defines two properties attached to the Event objects, namely the title of an event and the discipline related to. A URI and a label are associated to these properties (same process that the one for OWL classes); the range element defines the class pointed by the properties (here a gospl:Discipline or a gospl:Title); the domain element specify the source of the property. The inverseOf element allows to define the reciprocal property (for example, if we have child of property, owl:inverseOf allows to define that the reciprocal property is parent of). That very property makes the mechanism of logical inference possible.

The central class of the ontology is the Event class. The ontology specifies a huge amount subclasses of events, such as activity, audition, concert and debate among others and a lot of event properties, such as an event title, the event phone number, ... The ontology defines the concept of 'institution' in the same way. An institution can have a phone number, an address, a description, ...

To summarize, the ontology is composed of 219 entries defining the different concepts the cultural actors have in common, which makes the use of it primordial. Thanks to it, we can publish all the data from the participating actors in a unique dataset and making this data from heterogeneous sources linked.

3.3 Requirements

Given the State of the Art, the use cases and the cultural ontology, we can define the different requirement for the implementation.

The implementation must:

- be able to translate relational data from SQL databases into RDF;
- be able to translate relational data from XML databases into RDF;
- be able to map BOZAR semantized tables to the appropriate concepts;
- be open-source;
- output linked data;
- target the cultural ontology GOSPL providing a common vocabulary for linked data in the cultural domain;
- offer a command-line interface;
- be maintainable during the entire project;
The implementation should:

- translate data sources rapidly;
- implement a standard;
- manage the translation of several data sources at the same time;
- enrich data through call to external services (locations with Google Map service, ...).

The implementation can:

- offer a graphical user-interface;
- provide several methods of RDF serialization (turtle, XML, ...).

The most restrictive requirement is undoubtedly the possibility to translate the XML data sources from Agenda.be. As we have seen in the state of the art, it does not exist an implementation or a standard allowing to give XML trees as input and produce RDF graphs. Therefore, regardless the chosen implementation or standard, it will have to be modified.

This condition eliminates *ex officio* Triplify and Virtuoso, since the former is based on SQL and the latter is a too large solution (universal server) to be easily modified in order to accept XML.

Therefore, it remains a choice between use either an existing implementation, namely D2R Server or db2triples, or implement one (or both) of the current W3C standards. Direct Mapping is sadly too simple for the complexity provided by the use cases; in particular, the translation into the same vocabulary is impossible, since the metadata of both use cases are different.

D2R Server has a major quality that cannot be found in the other implementations: the conditional mapping. This functionality is particularly suited for mapping the BOZAR database that presents tables with metadata as data (values in the tuples that should be part of the schema instead). However, since the management of the data sources is integrated into the mapping language, modifying this language in order to accept XML source would entail a large amount of modifications in the whole implementations, without any guarantee of success (major issue). Moreover, those changes would not be reflected in a new release of D2R Server (minor issue).

Finally, we decided to implement R2RML rather than use db2triples or D2R Server. The main reason is that db2triples implements an outdated version of R2RML and that changing this implementation to match the last specification would represent as much work as creating a brand new one, since the standard encountered a consequent amount of fundamental modifications. We prefer to implement R2RML instead of D2RQ mainly because the latter contains a lot of words in its vocabulary, while R2RML is concise and leaves the data source management out of the mapping language. Adding new sources types is hence eased thanks to a clear separation between mapping language and implementation.
Chapter 4

Current Implementation

In this chapter we will approach the actual implementation of R2RML, written in the Java language. Secondly, we will detail the modification that have been applied to R2RML to match all the use cases. Finally, we will present the results of the translation obtained with the implementation and some performance analysis.

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Implementation design &amp; architecture</td>
<td>49</td>
</tr>
<tr>
<td>4.2 R2RML extensions</td>
<td>50</td>
</tr>
<tr>
<td>4.3 Results</td>
<td>54</td>
</tr>
</tbody>
</table>

4.1 Implementation design & architecture

The implementation has been written in Java for several reason. First, this is the language we master the best; we used it to waste the minimum of time to write the source code. Second, Java is world-wide used and possesses a large community of developer which facilitate the access to information and code examples. Finally, the main libraries that are used in this implementation (Jena and Saxon) were existing in Java.

This solution is divided in four packages.

The exceptions contains the different implementation of exceptions that could rise during the runtime.

The view offers place for classes composing the interface. For now, we provide two interfaces; the first one is a command-line tool that can easily be used into terminals; the second one is a graphical user interface allowing to select all the data sources and their associated mapping file and then dump them into a file with a chosen format while providing a debug functionality.

The controller provides the entry point of the application. It is charged to call the different classes and bind the view and the model.

The model contains the logic of the implementation. A UML chart with the interactions between the different components can be found in annex of this report. Basically, most of the
classes are straightforward implementation of the R2RML specification\[5\]. We can outline the implementation of an inheritance system for the logical tables since we have 4 types of them (from SQL table, SQL query, R2RML view and XPath query).

We have also created MappingFactory.java, whose role is the parsing of the turtle mapping file. It transform it into a jena graph and explore then the different nodes and create the corresponding object if necessary (logical tables, predicate-, object-, triples maps,...). As a result we have an array of TriplesMap objects. The different words offered by the R2RML vocabulary (and the extensions) are hard-coded in a constant hash map.

The triples created by the different triples maps are stored in a Jena graph, which is serialized at the end of the translation process.

**Libraries** we have to use three different libraries to make our solution working:

- mysql-jdbc-connector\[1\]: needed to access SQL databases
- Apache Jena API\[2\]: a framework to manipulate linked data.
- Saxon HE\[3\]: an implementation of XSLT, XQuery and XPath.

**The algorithm** for generating triples is simple and detailed in the pseudo-code (Algorithm\[1\]).

4.2 R2RML extensions

As a reminder, the use cases present two types of relational data sources candidate for a translation into linked open data. The BOZAR institution provides a relational SQL database while Agenda.be published its data in two XML documents, namely one describing the different events and the other the different institutions involved in those events.

\[1\] http://dev.mysql.com/downloads/connector/j/
\[2\] http://jena.apache.org/
\[3\] http://saxon.sourceforge.net/
Algorithm 1 Main algorithm to generate triples with the Triples Maps

TriplesMaps ← Parse mapping file
for all triples-map in TriplesMaps do
    logical-table ← triples-map
    subject-map ← triples-map
    for all tuple in logical-table do
        subject ← Generate-Subject(tuple, logical-table-schema)
        for all iri-class in subject-map do
            triple ← Generate-Subject-Class(subject, iri-class)
            Add triple into Graph
        end for
    end for
    for all predicate-object-map in triples-map do
        triple ← Generate-Predicate-Object-Pair(subject, tuple, logical-table-schema)
        Add triple into Graph
    end for
end for

The first issue that requires to be tackled is the multiplicity of input source types in order to be able to translate XML data. The second one is about the management of already triplified relational tables (tables where there are three columns for the subject, the predicate and the object).

4.2.1 XML sources translation

Stricto sensu, R2RML is designed to receive relational SQL databases as input. As a reminder, three mechanisms are provided to build the logical tables: (a) a existing table name; (b) a SQL query; and (c) a R2RML view. Nothing is planned to cope with XML documents.

The first solution to solve this issue is to create a small module, independent of the rest of the application, whose purpose is to un-nest the XML document and transform it into a table in respect with the format of other normal tables in the application. The element nodes used to form the column names and text nodes the different values in the table.

```xml
<?xml version="0.1" encoding="utf-8"?>
<documents>
  <document>
    <author>John Doe</author>
    <journal>New Kind of Paper</journal>
    <title>From RDB to RDF</title>
  </document>
</documents>
```

Listing 4.1: Simple XML document

<table>
<thead>
<tr>
<th>author</th>
<th>journal</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td>New Kind of Paper</td>
<td>From RDB to RDF</td>
</tr>
</tbody>
</table>

Table 4.1: Translation of a simple XML document into table
However, even if this solution works well for XML documents with a simple structure like a tree with a root and only one level of children (as shown in List 4.1), it is a bit too rough and ready to work with complex structures, like those present in the Agenda.be data.

The second solution is to enrich the R2RML specification with special words to take XML tables in account. This solution implies to modify the standard, which should be avoided if possible, since each mapping has to run on every implementation of R2RML. Nevertheless, the modification proposed will not lead this implementation to reject other mappings, since it is a simple addition in the vocabulary.

The idea is to allow the user to give a list of XPath expressions. The first element of this list gives the context node for all the following expressions. The table is built following these steps:

1. a set of nodes $\Gamma$ is created by applying the first expression of the list on the XML document.
2. the schema is defined by applying all the XPath expressions but the first on the first node of $\Gamma$ once. Each XPath expressions of the list gives an element node whose name becomes one of the column names. We assume that each XPath expression (except the first) returns one only one element node.
3. the table is filled by applying all the XPath expressions of the list but the first one on each node of $\Gamma$. We assume that each XPath expression (except the first) returns one only one element node. If is not the case, only the first node of the resulting set is taken in account.

The latter solution has been chosen in the implementation because it is much easier to implement than the former. Moreover, it let the user the possibility to select the nodes he is interested in.

We chose to add the resource `xpathQuery` in the R2RML namespace. As `rr:sqlQuery`, it is used as predicate in the `rr:logicalTable` blank node and it takes a string literal as associated object. This string will contain a set of valid XPath expressions separated by a separation character (we chose to use the `~` since it is forbidden in XPath expressions). If we look back to the previous XML code snippet, the XPath expression list would look like the following example.

```xml
<rr:xpathQuery>"///document~
  .//author~
  .//journal~
  .//title"
</rr:xpathQuery>
```

The resulting logical table is the same as table 4.1.

### 4.2.2 Semantized SQL tables

Since R2RML does not allow conditional mapping (that is, resources cannot be created in function of the value in a tuple), some tables of the BOZAR SQL database cause problems. Indeed, we can find some relations with a particular schema composed of four columns: `id`, `lng`, `field` and `content`. Actually, it can be considered as a representation of triples in a relational database. The `id` stands for the subject part, the field for the predicates and the content for the objects.
However, this representation is very not well suited for R2RML, because the only way to generate predicates are to give a constant (which is not sufficient to describe a table in most of the cases), a column name (but field does not mean anything) or a string template (but values in the field column are completely user-defined and heterogeneous so it is impossible to give a consistent string template).

Again there are two solutions to solve the problem due these semantized table:

1. Hard-code a predefined mapping in a file that bind the values in field to a proper URI. Then, while reading the particular semantized table, verify if the value being read belongs to blacklist; if it is the case, the generated resource is the one specified in the file. The main advantage is that the R2RML specification remains unmodified, since everything is at the implementation-level. The drawbacks are (a) the blacklist has to be maintained and completed each time a new type of value appears in the field column; (b) the user does not have the choice of the mapping since it is specified in the blacklist.

2. Transform the table during the logical table creation process in order to have a normal schema constituted by the distinct values from the field column. This idea is resumed on the schema below (Fig 4.2. The advantage of this method is that it allows the user to define its own mapping as he knows the name of the different values from field. The drawbacks are (a) the transformation process takes time, especially if the original table has a large number of tuples; (b) the R2RML specification is slightly modified with the adding of a new word.

![Figure 4.2: This picture describes the mechanism used to transform a semantized table into a usable logical table.](image)

We have chosen to implement the second solution, since its drawbacks are less restrictive. Furthermore, it better respects the purpose of R2RML which is to allow user to definer their own mappings.

We added the word semantized in the R2RML namespace. This predicate (rr:semantized) takes place in the blank node defining the rr:logicalTable of the triples map. It is associated with the name of the semantized table from the database. While parsing the mapping file, is this word is encountered, we create a new logical table by applying the following process:

- construction of the schema with the distinct values of the field column;
- filling of the table (see Algorithm 2, page 54).
Algorithm 2  Filling of the normalized Logical Table

\begin{algorithm}
\begin{algorithmic}
\ForAll{tuple in table}
\State id \leftarrow \text{table.get}(id) \hspace{1em} \triangleright \text{id used to check if needing to create a new tuple or use existing one}
\If{new-table.get(id) \neq \text{null}}
\State new-tuple \leftarrow \text{new-table.get}(id)
\Else
\State new-tuple \leftarrow \text{new Tuple()}
\State \text{add new-tuple in new-table}
\EndIf
\ForAll{column-name}
\If{column-name == content}
\State index \leftarrow \text{schema.getIndexOf(table.getValueIn(field))}
\State value \leftarrow \text{table.getValueIn(content)}
\ElseIf{column-name \neq field}
\State index \leftarrow \text{schema.getIndexOf(column-name)}
\State value \leftarrow \text{table.getValueIn(column-name)}
\EndIf
\State new-tuple[index] \leftarrow value
\EndFor
\EndFor
\end{algorithmic}
\end{algorithm}

4.3 Results

4.3.1 Mapping to the ontologies

An important part of the work has been the mapping to the different concepts to the afore-described ontology. A major issue was that we had to manage two different data sources that are globally dealing with the same concepts (e.g., events) but they are using different schemas and taxonomies to designate same things. For instance, the concept of event is clearly named Event_Instance in the Agenda.be data while BOZAR uses the word activity.

In the following tables, we propose a possible mapping for both use cases. The first column of those table contains the column name of the BOZAR table or the element node of the Agenda.be XML tree and the second column contains the associated concept from the gospl ontology.

<table>
<thead>
<tr>
<th>Column</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>activity.season</td>
<td>bozar:season</td>
</tr>
<tr>
<td>activity.date_end</td>
<td>gospl:DateTimeSpecification_valid_until_Date</td>
</tr>
<tr>
<td>activity.date_start</td>
<td>gospl:DateTimeSpecification_valid_from_Date</td>
</tr>
<tr>
<td>activity.title_int</td>
<td>gospl:Event_has_Title</td>
</tr>
<tr>
<td>activity_cible.id_cible</td>
<td>gospl:Event_with_Target</td>
</tr>
<tr>
<td>activity_language.id_language</td>
<td>gospl:Event_in_Language</td>
</tr>
<tr>
<td>activity_category.id_category</td>
<td>gospl:Event_with_Category</td>
</tr>
<tr>
<td>activity_category</td>
<td>gospl:Category</td>
</tr>
<tr>
<td>activity_category.name_fr</td>
<td>gospl:Category_with_Value</td>
</tr>
<tr>
<td>activity_category.name_nl</td>
<td>gospl:Category_with_Value</td>
</tr>
<tr>
<td>activity_category.name_en</td>
<td>gospl:Category_with_Value</td>
</tr>
<tr>
<td>activity_lng.field.title</td>
<td>gospl:Event_has_Title</td>
</tr>
<tr>
<td>location_lng.field.zip</td>
<td>gospl:Address_with_Zip_Code</td>
</tr>
</tbody>
</table>
Table 4.2: Mapping of the BOZAR database to the gospl ontology.

<table>
<thead>
<tr>
<th>Element node</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventID</td>
<td>gospl:Event</td>
</tr>
<tr>
<td>Event_Title_FR</td>
<td>gospl:Event_has_Title</td>
</tr>
<tr>
<td>Event_Title_NL</td>
<td>gospl:Event_has_Title</td>
</tr>
<tr>
<td>Event_Title_EN</td>
<td>gospl:Event_has_Title</td>
</tr>
<tr>
<td>Event_ShortDescription_FR</td>
<td>gospl:Event_with_Description</td>
</tr>
<tr>
<td>Event_ShortDescription_NL</td>
<td>gospl:Event_with_Description</td>
</tr>
<tr>
<td>Event_ShortDescription_EN</td>
<td>gospl:Event_with_Description</td>
</tr>
<tr>
<td>Event_LongDescription_FR</td>
<td>gospl:Event_with_Description</td>
</tr>
<tr>
<td>Event_LongDescription_NL</td>
<td>gospl:Event_with_Description</td>
</tr>
<tr>
<td>Event_LongDescription_EN</td>
<td>gospl:Event_with_Description</td>
</tr>
<tr>
<td>Event_ContactPhone</td>
<td>gospl:Event_with_contact_Phone</td>
</tr>
<tr>
<td>Event_BookingPhone</td>
<td>gospl:Event_with_Booking_Phone</td>
</tr>
<tr>
<td>Event_Fax</td>
<td>gospl:Event_with_fax_Phone</td>
</tr>
<tr>
<td>Event_Mail_FR</td>
<td>gospl:Event_with_Email_Address</td>
</tr>
<tr>
<td>Event_Mail_NL</td>
<td>gospl:Event_with_Email_Address</td>
</tr>
<tr>
<td>Event_Price_FR</td>
<td>gospl:Event_with_Price</td>
</tr>
<tr>
<td>Event_Price_NL</td>
<td>gospl:Event_with_Price</td>
</tr>
<tr>
<td>Event_Price_Description_FR</td>
<td>gospl:Event_with_Price_Description</td>
</tr>
<tr>
<td>Event_Price_Description_NL</td>
<td>gospl:Event_with_Price_Description</td>
</tr>
<tr>
<td>Event_Price_Description_EN</td>
<td>gospl:Event_with_Price_Description</td>
</tr>
<tr>
<td>Event-LanguageDescription_FR</td>
<td>gospl:Event_in_Language</td>
</tr>
<tr>
<td>Event Language Description NL</td>
<td>gospl:Event_in_Language</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Event_Canceled</td>
<td>gospl:Event_with_Cancel_Status</td>
</tr>
<tr>
<td>Event_SoldOut</td>
<td>gospl:Event_with_Sold_Out_Status</td>
</tr>
<tr>
<td>Event_CategoryDescription_FR</td>
<td>gospl:Event_with_Category</td>
</tr>
<tr>
<td>Event_CategoryDescription_NL</td>
<td>gospl:Event_with_Category</td>
</tr>
<tr>
<td>Event_GroupDescription_FR</td>
<td>gospl:Event_with_Group</td>
</tr>
<tr>
<td>Event_GroupDescription_NL</td>
<td>gospl:Event_with_Group</td>
</tr>
<tr>
<td>Event_Scope_FR</td>
<td>gospl:Event_with_Scope</td>
</tr>
<tr>
<td>Event_Scope_NL</td>
<td>gospl:Event_with_Scope</td>
</tr>
<tr>
<td>Event_Performer_FR</td>
<td>gospl:Event_with_Performer</td>
</tr>
<tr>
<td>Event_Performer_NL</td>
<td>gospl:Event_with_Performer</td>
</tr>
<tr>
<td>Event_Performer_EN</td>
<td>gospl:Event_with_Performer</td>
</tr>
<tr>
<td>Event_DateFrom</td>
<td>gospl:DateTimeSpecification_valid_from_Date</td>
</tr>
<tr>
<td>Event_DateTo</td>
<td>gospl:DateTimeSpecification_valid_until_Date</td>
</tr>
<tr>
<td>Event_Partenaire_1</td>
<td>gospl:Event_with_Partner</td>
</tr>
<tr>
<td>Event_Linked/Event_Linked_EventID</td>
<td>gospl:Event_linked_with_Event</td>
</tr>
<tr>
<td>Event_Booker_1/Event_Booker_InstitutionID</td>
<td>gospl:Event_with_booker_institution</td>
</tr>
</tbody>
</table>

Table 4.3: Mapping of the Agenda.be database to the gospl ontology. The used prefix used gospl for [http://starpc18.vub.ac.be:8080/gospl/ontology/2#](http://starpc18.vub.ac.be:8080/gospl/ontology/2#).

We give here two R2RML predicate-object maps that allows to generate the title of events from both data sources, using the mapping tables we just defined:

```rml
rr: predicateObjectMap [
    rr: predicate gospl: Event_with_Title;
    rr: objectMap [
        rr: column "title_int";
        rr: datatype xsd:string;
        rr: termType rr: Literal;
    ]
];
```

Listing 4.2: R2RML predicate-object map for generating event titles in BOZAR

```rml
rr: predicateObjectMap [
    rr: predicate gospl: Event_has_Title;
    rr: objectMap [
        rr: column "Event_Title_FR" ;
        rr: language "fr-BE";
        rr: termType rr: Literal;
    ];
];
```

Listing 4.3: R2RML predicate-object map for generating event titles in Agenda.be

The complete mappings written with the valid R2RML syntax are given in the appendices. Nonetheless, some parts of those mappings can be detailed, e.g. the logical table creation for the BOZAR activity table:

```rml
rr: logicalTable [
    rr: semantizedTable "activity_lng";
    rr: sqlQuery ""
    SELECT a.id, a.season, a.date_end, a.date_start, a.title_int, a.status_canceled, a.statusCanceled,
    ∨, a.status_canceled, a.statusCanceled,
| 56 |
The logical table is an aggregation of several tables dealing with activities: _activity_cible, _activity_language, _activity_category and activity_lng. The left outer join operations are necessary to guarantee that each activity from the activity table will be represented. A simple join would have selected only the tuples whose id value being in every the 'activity-driven' tables. From this example we can point out that the performer of the mapping can use all the SQL functionalities for query since the query execution is left to the SQL driver. The only constraint remains that the result of the query must be a single table.

4.3.2 Ontologies enrichment

Since we have deeply dived into the data of the different cultural actors, we have been able to enrich the ontology with concepts we used to judge useful and that have considerably increased the knowledge carried by the data published in RDF.

We have provided two addenda to the ontology developed by Star Lab. The first describes musical opuses, which are mainly present in the BOZAR database, and the second cultural-driven locations. The concepts concerning opuses are the following:

<table>
<thead>
<tr>
<th>HEAD</th>
<th>→</th>
<th>←</th>
<th>TAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opus</td>
<td>created at</td>
<td>of creation of</td>
<td>Location</td>
</tr>
<tr>
<td>Opus</td>
<td>created on</td>
<td>of creation of</td>
<td>Date</td>
</tr>
<tr>
<td>Opus</td>
<td>with category</td>
<td>of category of</td>
<td>Music</td>
</tr>
<tr>
<td>Opus</td>
<td>with</td>
<td>of</td>
<td>Duration</td>
</tr>
<tr>
<td>Opus</td>
<td>with</td>
<td>of</td>
<td>Subtitle</td>
</tr>
<tr>
<td>Opus</td>
<td>with</td>
<td>of</td>
<td>Title</td>
</tr>
<tr>
<td>Opus</td>
<td>with</td>
<td>of</td>
<td>Tone</td>
</tr>
<tr>
<td>Opus</td>
<td>with</td>
<td>of</td>
<td>Author</td>
</tr>
</tbody>
</table>

Table 4.4: Completion of the cultural domain ontology with concepts describing musical opuses

We decided to add concepts describing locations as well in order to enrich the linked data with geographical data (coordinates, latitude, longitude,...). Those kind of data could be very
useful for creating mobile applications, for example, that can be used to query information in relation with the geographical location of the user that are stored in the dataset (for instance, "I want to find all the museums in radius of 2 miles around my position").

The concepts we propose are the following:

<table>
<thead>
<tr>
<th>HEAD</th>
<th>→</th>
<th>←</th>
<th>TAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>has of</td>
<td>Coordinates</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>has of</td>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Coordinates</td>
<td>with of</td>
<td>Longitude</td>
<td></td>
</tr>
<tr>
<td>Coordinates</td>
<td>with of</td>
<td>Latitude</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>with of</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>with of</td>
<td>street</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>with of</td>
<td>Zipcode</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>with of</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>subsumes is a</td>
<td>Private Area</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>subsumes is a</td>
<td>Public Area</td>
<td></td>
</tr>
<tr>
<td>Private Area</td>
<td>subsumes is a</td>
<td>House</td>
<td></td>
</tr>
<tr>
<td>Private Area</td>
<td>subsumes is a</td>
<td>Office</td>
<td></td>
</tr>
<tr>
<td>Private Area</td>
<td>subsumes is a</td>
<td>Apartment</td>
<td></td>
</tr>
<tr>
<td>Public Area</td>
<td>subsumes is a</td>
<td>Building</td>
<td></td>
</tr>
<tr>
<td>Public area</td>
<td>subsumes is a</td>
<td>Station</td>
<td></td>
</tr>
<tr>
<td>Public area</td>
<td>subsumes is a</td>
<td>Haven</td>
<td></td>
</tr>
<tr>
<td>Public area</td>
<td>subsumes is a</td>
<td>Airport</td>
<td></td>
</tr>
<tr>
<td>Public area</td>
<td>subsumes is a</td>
<td>Graveyard</td>
<td></td>
</tr>
<tr>
<td>Public area</td>
<td>subsumes is a</td>
<td>Oil station</td>
<td></td>
</tr>
<tr>
<td>Station</td>
<td>subsumes is a</td>
<td>Bus halt</td>
<td></td>
</tr>
<tr>
<td>Station</td>
<td>subsumes is a</td>
<td>Tram halt</td>
<td></td>
</tr>
<tr>
<td>Station</td>
<td>subsumes is a</td>
<td>Train station</td>
<td></td>
</tr>
<tr>
<td>Station</td>
<td>subsumes is a</td>
<td>Subway station</td>
<td></td>
</tr>
<tr>
<td>Building</td>
<td>subsumes is a</td>
<td>Bar</td>
<td></td>
</tr>
<tr>
<td>Building</td>
<td>subsumes is a</td>
<td>Restaurant</td>
<td></td>
</tr>
<tr>
<td>Building</td>
<td>subsumes is a</td>
<td>Museum</td>
<td></td>
</tr>
<tr>
<td>Building</td>
<td>subsumes is a</td>
<td>Hostel</td>
<td></td>
</tr>
<tr>
<td>Building</td>
<td>subsumes is a</td>
<td>Hospital</td>
<td></td>
</tr>
<tr>
<td>Building</td>
<td>subsumes is a</td>
<td>Store</td>
<td></td>
</tr>
<tr>
<td>Building</td>
<td>subsumes is a</td>
<td>Teaching place</td>
<td></td>
</tr>
<tr>
<td>Building</td>
<td>subsumes is a</td>
<td>Place of worship</td>
<td></td>
</tr>
<tr>
<td>Place of worship</td>
<td>subsumes is a</td>
<td>Church</td>
<td></td>
</tr>
<tr>
<td>Place of worship</td>
<td>subsumes is a</td>
<td>Mosque</td>
<td></td>
</tr>
<tr>
<td>Place of worship</td>
<td>subsumes is a</td>
<td>Synagogue</td>
<td></td>
</tr>
<tr>
<td>Place of worship</td>
<td>subsumes is a</td>
<td>Temple</td>
<td></td>
</tr>
<tr>
<td>Teaching place</td>
<td>subsumes is a</td>
<td>School</td>
<td></td>
</tr>
<tr>
<td>Teaching place</td>
<td>subsumes is a</td>
<td>University</td>
<td></td>
</tr>
<tr>
<td>Teaching place</td>
<td>subsumes is a</td>
<td>Art Schools</td>
<td></td>
</tr>
</tbody>
</table>

*Table 4.5: Location ontology describing cultural places*
4.3.3 Translation of the relational data

The complete dataset resulting of the translation of the databases can be found on the Mercurial server of the laboratory supporting the thesis. In the following listing, we are presenting a small part of the triples translated from the BOZAR database:

```xml
<http://bozar/db/activity/9151>
a
<http://starpc18.vub.ac.be:8080/gospl/ontology/2#event> ;
<http://bozar.be/db#ID>
"9151"^^<http://www.w3.org/2001/XMLSchema#decimal> ;
<http://bozar.be/db#season>
"2009"^^<http://www.w3.org/2001/XMLSchema#gYear> ;
<http://starpc18.vub.ac.be:8080/gospl/ontology/2#>
\ DateTimeSpecification_valid_from_Date>
"2009-10-25"^^<http://www.w3.org/2001/XMLSchema#date> ;
<http://starpc18.vub.ac.be:8080/gospl/ontology/2#>
\ DateTimeSpecification_valid_until_Date>
"2009-10-25"^^<http://www.w3.org/2001/XMLSchema#date> ;
<http://starpc18.vub.ac.be:8080/gospl/ontology/2#Title>
"BOZAR SUNDAYS Good Morning, Koen Plaetinck 25/10/09"^^<http://www.w3.org/2001/XMLSchema#string> .
```

Listing 4.5: Partial result of the translation of the tables activity and activity_ing from the BOZAR database. Turtle notation.

On the previous result set, we can see an activity translated into RDF. Lines 1 and 2 describes the subject and the type of the resource while other lines are predicate-object couples. We can point out that all the concepts used come from the ontology defining the cultural things (http://starpc18.vub.ac.be:8080/gospl/ontology/2#). The different data represented are from the tables activity and activity_ing. We can also remark that the implementation manages literal datatyping.

On the following listings, we have a partial result of the translation of the agenda.be XML file dealing with events:

```xml
<Event_Instance>
<EventID>234399</EventID>
<Event_Title_FR>Andrea Croonenberghs, Marc Dex en Jan Mues,</Event_Title_FR>
<Event_Title_NL>Andrea Croonenberghs, Marc Dex en Jan Mues,</Event_Title_NL>

<Event_Canceled>False</Event_Canceled>
</Event_Instance>
```

Listing 4.6: Slice of an event node of the original 'event' XML file.
Table 4.6: Average results of the running of the various mappings on their respective data sources. TM stands for Triples Maps and LT for Logical Table. POM stands for Predicate-Object Map.

<table>
<thead>
<tr>
<th></th>
<th>Translation time (ms)</th>
<th>LT building time (ms)</th>
<th># tuples</th>
<th>Schema size</th>
<th># POM</th>
</tr>
</thead>
<tbody>
<tr>
<td>opus lng</td>
<td>573</td>
<td>722</td>
<td>5861</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>location lng</td>
<td>3</td>
<td>12</td>
<td>64</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>personnality lng</td>
<td>177</td>
<td>230</td>
<td>7739</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>activity lng</td>
<td>1452</td>
<td>4142</td>
<td>5853</td>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>cible</td>
<td>7</td>
<td>227</td>
<td>60</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>activity category</td>
<td>2</td>
<td>2</td>
<td>57</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>language</td>
<td>18</td>
<td>6</td>
<td>461</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Event_Instance</td>
<td>971</td>
<td>1292</td>
<td>422</td>
<td>38</td>
<td>37</td>
</tr>
<tr>
<td>Institution_Instance</td>
<td>135</td>
<td>771</td>
<td>121</td>
<td>26</td>
<td>25</td>
</tr>
</tbody>
</table>

On this example, we have shown that the translation worked for XML data sets as well. The various data are mapped to the same ontology (gospl); resulting linked data from both relational data sources belong hence to the same dataset. We can outline that language tags are managed as well.

The resulting dataset present enough information about events (such as location, opening/closing hours, type of event, targeted people and so on) to be used in application whose purpose is the manipulation of those data, for example mobile applications. Even if the mappings could eventually be completed in some way or another, they already make the translation of most of the data common to the various cultural actors possible.

4.3.4 Performance analysis

In order to analyse the different parts of the algorithm that might be improved, we have run the different mappings on the data sources and measured the time of translation (application of the triples maps) and the time of logical table creation. The following table contains the average values of the three measurements.

From this different values, we can draw the following charts, representing namely i) the time to construct the logical table; ii) the global speed of translation in function of the schema size; and iii) the total translation time in function of the schema size.
In the previous figure, wherein the blue lines represent SQL data and the orange XML data, it clearly appears that our implementation is strongly dependant of the size of the schema for the semantized tables. This can easily be explained by the fact that we have to transform the SQL table by checking the field value of the tuple and place the corresponding content value at the right place in the new tuples. Since the size of the schema is given by the distinct values of the field attribute, the more fields we have, the more tests we have to make.

We assume that the process is faster for XML data because once the tree is in memory, the different XPath expressions are fastly executed with the Saxon library. However, since we do not have enough XML data samples, we cannot deduce a global trend for this kind of data.
The above figure indicates the speed of translation (tuples per second) in function of the schema size. The disparity of the measures for small schemas is explained that in the use cases the tables with small schemas contain a little amount of tuples, that were treated in a short time, close to the smallest unit of time (ms) of the measurement device. We obtained hence variations between the different experiences. Nevertheless, we can outline a exponential trend inversely proportional to the size of the schema, as shown in the figure.

![Total Translation Time in function of the logical table schema size](image)

**Figure 4.5:** Total translation time

Finally, the global translation time (application of the triples maps) seems to be directly proportional to the schema size.

To summarize those different measures, we can conclude that the mappings should present optimized queries that return a table with the smallest schema, knowing that increasing its size directly influences the logical table creation time, the translation time and the speed of translation.

The data in our possession does not contain enough tuples allowing us to determine an exact correlation between the them and the runtime, even though we can assume that increasing the number of tuples evidently increases the time of translation.
Chapter 5

Conclusion

In this thesis, we have presented our work concerning the relational-to-RDF translation issue. We have begun this journey by reminding the different concepts needed to clearly understand the problem, namely the definition of the Semantic Web, a short presentation of the URI system, the Resource Description Framework and the Linking Open Data project.

This led us to present the use cases if this memoir. The first one is the translation of the BOZAR relational SQL database. The second one is the translation of the two Agenda.be XML files, composed of the events and the institutions.

To address the different needs of these use cases, we have compiled a state of art concerning the problem. In this state of the art, we have presented the major solutions such as OpenLink Virtuoso Universal Server, D2R Server, db2triples and Triplify and the two standards under development by the W3C, Direct Mapping (DM) and Relational-to-RDF Mapping Language (R2RML). We concluded this state of the art by comparing the various strengths and weaknesses of each of its aforementioned components. From this comparison, we have observed a lack in the state of the art and hence decided to create our own implementation of R2RML, allowing us to slightly alter the standard.

Then we have briefly detailed the different parts of our implementation, we have presented the ontologies that were used to map the different concepts encountered in the two use case databases and we have described the changes we brought to the R2RML standard, namely the possibilities to have XML data sources as input and to transform semantized tables.

We have concluded this thesis by reporting the different encouraging results of the application of the implementation to the use cases and what we have retained of this peculiar experience.

In a close future, the prevailing project is the implementation of a module in the application whose role would be the enrichment of the location data. For now, relational data does not store geographical information (longitude, latitude,...). However, some services, like Google Maps, provide the functionality to retrieve those data from place names. Having these geographical data would make possible the use of the cultural dataset to mobile applications browsing semantic Web (for instance, find the other museums in this area).

Some work can be done to improve local pieces of the algorithm, for instance the normalization of normal table, which strongly depends on the number of tuples in the table (non-linear complexity).
Finally, the different ontologies used in the translation process can be completed in order to give the possibility to users to perform better mappings on their databases and improve the quality of the linked data.
Appendices
Appendix A

Introduction: BOZAR tables

DATABASE :
bozar_v01
TABLES :
** activity
database_copy
database_lng
database
cible
highlight_lng
** language
** location
news
newstypes
** opus
part
** personnality
personnality_function
picturefile
psection
psectionitem
relation
relation_lng
salle_lng
schedule
schedule_price
sponsor
webdoc
webdoc_lng
webpage
_.activity._activity
_.activity._cible
_.activity._language
_.activity._section
_.activity._section
_.news._activity
_.news._type
_.part._cible
_.part._personnality
** activity_category
database_keywords
database_type
highlight
database_keywords
language_function
** location_lng
newsletter_section
news_lng
** opus_lng
part_lng
personnality_category
** personnality_lng
psection
psectionparent
relation_function
salle
salle_url
schedule_lng
section
sponsor_lng
webdoc_link
webfile
webpage_lng
** _.activity._category
_.activity._favorites
_.activity._relation
_.activity._type
_.highlight._section
_.news._section
** _.opus._personnality
_.part._opus
_.personnality._category
Appendix B

State of the Art: Results of the running example mappings

```ttl
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ulb: <http://ulb.ac.be/db/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

5<http://ulb.ac.be/db/laboratory/1> rdf:type <http://ulb.ac.be/db/Laboratory> ;
   ulb:lname "Web and Information Technologies" ;
   ulb:location "Building A.5" ;
   ulb:directedBy <http://ulb.ac.be/db/researcher/5> .
9<http://ulb.ac.be/db/laboratory/2> rdf:type <http://ulb.ac.be/db/Laboratory> ;
   ulb:lname "Operational research" ;
   ulb:location "Building A.3" ;
13<http://ulb.ac.be/db/laboratory/3> rdf:type <http://ulb.ac.be/db/Laboratory> ;
   ulb:lname "Artificial Intelligence" ;
   ulb:location "Building F.6" ;

18<http://ulb.ac.be/db/researcher/1> rdf:type <http://ulb.ac.be/db/Researcher> ;
   foaf:name "Doe" ;
   ulb:worksIn <http://ulb.ac.be/db/laboratory/1> .
   foaf:name "Will" ;
   ulb:worksIn <http://ulb.ac.be/db/laboratory/1> .
   foaf:name "Robert" ;
   ulb:worksIn <http://ulb.ac.be/db/laboratory/2> .
   foaf:name "Smith" ;
   ulb:worksIn <http://ulb.ac.be/db/laboratory/3> .
30<http://ulb.ac.be/db/researcher/5> rdf:type <http://ulb.ac.be/db/Researcher> ;
   foaf:name "Jack" ;
   ulb:worksIn <http://ulb.ac.be/db/laboratory/1> .
```
Listing B.1: Virtuoso RDF View of the running example

```xml
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ulb: <http://ulb.ac.be/db/> .

<http://ulb.ac.be/db/laboratory/1> rdf:type ulb:Laboratory ;
ulb:directedBy <http://ulb.ac.be/db/researcher/5> .
<http://ulb.ac.be/db/laboratory/2> rdf:type ulb:Laboratory ;
<http://ulb.ac.be/db/laboratory/3> rdf:type ulb:Laboratory ;

<http://ulb.ac.be/db/researcher/1> rdf:type <http://ulb.ac.be/db/researcher> ;
foaf:name "Doe"^^xsd:string ;
ulb:worksIn <http://ulb.ac.be/db/laboratory/1> .
<http://ulb.ac.be/db/researcher/2> rdf:type <http://ulb.ac.be/db/researcher> ;
foaf:name "Will"^^xsd:string ;
ulb:worksIn <http://ulb.ac.be/db/laboratory/1> .
<http://ulb.ac.be/db/researcher/3> rdf:type <http://ulb.ac.be/db/researcher> ;
foaf:name "Robert"^^xsd:string ;
ulb:worksIn <http://ulb.ac.be/db/laboratory/2> .
foaf:name "Smith"^^xsd:string ;
ulb:worksIn <http://ulb.ac.be/db/laboratory/3> .
<http://ulb.ac.be/db/researcher/5> rdf:type <http://ulb.ac.be/db/researcher> ;
foaf:name "Jack"^^xsd:string ;
ulb:worksIn <http://ulb.ac.be/db/laboratory/1> .
<http://ulb.ac.be/db/researcher/6> rdf:type <http://ulb.ac.be/db/researcher> ;
foaf:name "Tom"^^xsd:string ;
ulb:worksIn <http://ulb.ac.be/db/laboratory/3> .
<http://ulb.ac.be/db/researcher/7> rdf:type <http://ulb.ac.be/db/researcher> ;
foaf:name "Andrea"^^xsd:string ;
ulb:worksIn <http://ulb.ac.be/db/laboratory/2> .
foaf:name "Alice"^^xsd:string ;
ulb:worksIn <http://ulb.ac.be/db/laboratory/3> .
```
foaf:name "Sophie"^^xsd:string;
ulb:worksIn <http://ulb.ac.be/db/laboratory/2>.
foaf:name "Alice"^^xsd:string;
ulb:worksIn <http://ulb.ac.be/db/laboratory/3>.

Listing B.2: D2RQ mapping results

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ulb: <http://ulb.ac.be/db/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://ulb.ac.be/db/laboratory/1> rdf:type <http://ulb.ac.be/db/Laboratory> ;
  ulb:lid "1";
  ulb:lname "Web and Information Technologies"@en-GB ;
  ulb:directedBy <http://ulb.ac.be/db/researcher/5> .
<http://ulb.ac.be/db/laboratory/2> rdf:type <http://ulb.ac.be/db/Laboratory> ;
  ulb:lid "2";
  ulb:lname "Operational research"@en-GB ;
<http://ulb.ac.be/db/laboratory/3> rdf:type <http://ulb.ac.be/db/Laboratory> ;
  ulb:lid "3";
  ulb:lname "Artificial Intelligence"@en-GB ;

<http://ulb.ac.be/db/researcher/1> rdf:type <http://ulb.ac.be/db/Researcher> ;
  ulb:rid "1"^^xsd:integer;
  foaf:name "Doe"^^xsd:string;
  ulb:worksIn <http://ulb.ac.be/db/laboratory/1> .
  ulb:rid "2"^^xsd:integer;
  foaf:name "Will"^^xsd:string;
  ulb:worksIn <http://ulb.ac.be/db/laboratory/1> .
  ulb:rid "3"^^xsd:integer;
  foaf:name "Robert"^^xsd:string;
  ulb:worksIn <http://ulb.ac.be/db/laboratory/2> .
  ulb:rid "4"^^xsd:integer;
  foaf:name "Smith"^^xsd:string;
  ulb:worksIn <http://ulb.ac.be/db/laboratory/3> .
<http://ulb.ac.be/db/researcher/5> rdf:type <http://ulb.ac.be/db/Researcher> ;
  ulb:rid "5"^^xsd:integer;
  foaf:name "Jack"^^xsd:string;
  ulb:worksIn <http://ulb.ac.be/db/laboratory/1> .
  ulb:rid "6"^^xsd:integer;
  foaf:name "Tom"^^xsd:string;
  ulb:worksIn <http://ulb.ac.be/db/laboratory/3> .
Listing B.3: RDF dump of R2RML mapping applied on the running example
Appendix C

Implementation architecture
Appendix D

BOZAR mapping

```xml
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix gospl: <http://starpc18.vub.ac.be:8080/gospl/ontology/2#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix bozar: <http://bozar.be/db#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

#TM_activities
  a rr:TriplesMap;
  rr:logicalTable [
    rr:semantizedTable "activity_lng";
    rr:sqlQuery ""
      SELECT a.id, a.season, a.date_end, a.date_start, a.title_int,
        a.statusCanceled, a.statusCanceled,
        ac.id_cible as id_cible,
        al.id_language as id_language,
        acat.id_category as id_category,
        alng.field as field, alng.content as content
      FROM activity a
      left outer join _activity_cible ac ON ac.id_activity = a.
        id
      left outer join _activity_language al ON al.id_activity = a.
        id
      left outer join _activity_category acat ON acat.
        id_activity = a.id,
      activity_lng alng
      WHERE alng.id = a.id
    "";]

  rr:subjectMap [
    rr:template "http://bozar.be/db/activity/{id}";
    rr:class gospl:event
  ];

  rr:predicateObjectMap [
    rr:predicate bozar:ID;
    rr:objectMap [
      rr:column "id";
      rr:datatype xsd:decimal
    ]
  ];
```

rr: predicateObjectMap {
    rr: predicate bozar:season ;
    rr: objectMap [
        rr: column "season";
        rr: datatype xsd:gYear;
    ]
};

rr: predicateObjectMap [
    rr: predicate gospl:DateTimeSpecification_valid_until_Date;
    rr: objectMap [
        rr: column "date_end";
        rr: datatype xsd:date
    ];
];

rr: predicateObjectMap [
    rr: predicate gospl:DateTimeSpecification_valid_from_Date;
    rr: objectMap [
        rr: column "date_start";
        rr: datatype xsd:date
    ];
];

rr: predicateObjectMap [
    rr: predicate gospl:Title;
    rr: objectMap [
        rr: column "title_int";
        rr: datatype xsd:string
    ]
];

rr: predicateObjectMap [
    rr: predicate gospl:Event_with_Target;
    rr: refObjectMap [
        rr: parentTriplesMap <# TM_targets >;
        rr: joinCondition [
            rr: child "id_cible";
            rr: parent "id";
        ];
    ];
];

rr: predicateObjectMap [
    rr: predicate gospl:Event_in_Language;
    rr: refObjectMap [
        rr: parentTriplesMap <# TM_languages >;
        rr: joinCondition [
            rr: child "id_language";
            rr: parent "id";
        ];
    ];
];

rr: predicateObjectMap [
    rr: predicate gospl:Event_with_Category;
    rr: refObjectMap [
        rr: parentTriplesMap <# TM_activity-categories >;
        rr: joinCondition [
            rr: child "id_category";
            rr: parent "id";
        ];
    ];
];
<# TM_activity_categories >

a $rr$:TriplesMap ;

$rr$:tableName "activity_category";

$rr$:subjectMap [
  $rr$:template "http://bozar.be/db/Category/{id}";
  $rr$:class gospl:Category;
];

$rr$:predicateObjectMap [
  $rr$:predicate gospl:Category_with_Value;
  $rr$:objectMap [
    $rr$:column "name_fr";
    $rr$:language "fr-BE";
  ];
];

$rr$:predicateObjectMap [
  $rr$:predicate gospl:Category_with_Value;
  $rr$:objectMap [
    $rr$:column "name_nl";
    $rr$:language "nl-BE";
  ];
];

$rr$:predicateObjectMap [
  $rr$:predicate gospl:Category_with_Value;
  $rr$:objectMap [
    $rr$:column "name_en";
    $rr$:language "en-GB";
  ];
];

<# TM_locations >

a $rr$:TriplesMap ;

$rr$:logicalTable [
  $rr$:semantizedTable "location_lng";
  $rr$:sqlQuery ""
    SELECT distinct loc.id AS id,
    loc.active AS active,
    loclng.lng AS lng,
    loclng.field AS field,
    loclng.content AS content
    FROM location loc,
    location_lng loclng
    WHERE loclng.id = loc.id
  "";
];

$rr$:subjectMap [
  $rr$:template "http://bozar.be/db/Locations/{id}";
  $rr$:class gospl:Location;
];

## POSSIBLE FIELDS VALUES : name, city, country, address, zip

$rr$:predicateObjectMap [
  $rr$:predicate gospl:Address_with_Zip_Code;
  $rr$:objectMap [
    $rr$:column "zip";
  ];
]
<?xml version="1.0" encoding="UTF-8"?>
<#TM_personnalities>
    a rr:TriplesMap ;
    rr:logicalTable [ rr:semantizedTable "personnality_lng" ;
        rr:sqlQuery ""
        SELECT p.id AS id,
            p.id_language AS id_language,
            p.url0 AS url0,
            plng.lng AS lng,
            plng.field AS field,
            plng.content AS content
        FROM personnality p,
            personnality_lng plng
        WHERE p.id = plng.id
        "" ;
    ];
    rr:subjectMap [ rr:template "http://bozar.be/db/personnality/{id}" ;
        rr:class foaf:Person ;
    ];
    ## POSSIBLE FIELD VALUES: name, end_date, start_date, start_location, ...
    end_location, location
    rr:predicateObjectMap [ rr:predicate foaf:name ;
        rr:objectMap [ rr:column "name" ;
            rr:datatype xsd:string ;
        ];
    ];
</#TM_personnalities>
<#TM_targets>
a rr:TriplesMap ;
rr:tableName "cible";

rr:subjectMap [ rr:template "http://bozar.be/db/targets/{id}";
rr:className gospl:Target ; ];

rr:predicateObjectMap [ rr:predicate gospl:Target_with_Value ; rr:objectMap [ rr:column "name_fr";
rr:language "fr-BE";
rr:termType rr:Literal ; ]];

rr:predicateObjectMap [ rr:predicate gospl:Target_with_Value ; rr:objectMap [ rr:column "name_nl";
rr:language "nl-BE";
rr:termType rr:Literal ; ]];

rr:predicateObjectMap [ rr:predicate gospl:Target_with_Value ; rr:objectMap [ rr:column "name_en";
rr:language "en-GB";
rr:termType rr:Literal ; ]];

<#TM_opus>
a rr:TriplesMap ;

rr:logicalTable [ rr:semantizedTable "opus_lng";
rr:sqlQuery ""
SELECT distinct o.id AS id,
o.title_int AS title_int,
o.id_language AS id_language,
olng.lng AS lng,
olng.field AS field,
olng.content AS content,
op.id_personnality as id_personnality
FROM opus_lng olnng,
opus o,
_opus_personnality op
WHERE o.id = olnng.id
AND op.id_opus = o.id
"" ;
];
rr:subjectMap [  
  rr:template "http://bozar.be/db/Opus/{id}" ;  
  rr:class gospl:Opus ;  
];  

## POSSIBLE PREDICATES : title, duration, category, tone, /
\ creation_date, subtitle, elements, creation_locatio, type  
rr:predicateObjectMap [  
  rr:predicate gospl:Opus_with_Title ;  
  rr:objectMap [  
    rr:column "title_int";  
    rr:datatype xsd:string ;  
    rr:termType rr:Literal ;  
  ];  
];  
rr:predicateObjectMap [  
  rr:predicate gospl:Opus_with_Tone ;  
  rr:objectMap [  
    rr:column "tone";  
    rr:datatype xsd:string;  
    rr:termType rr:Literal;  
  ];  
];  
rr:predicateObjectMap [  
  rr:predicate gospl:Opus_with_Duration ;  
  rr:objectMap [  
    rr:column "duration";  
    rr:datatype xsd:string;  
    rr:termType rr:Literal;  
  ];  
];  
rr:predicateObjectMap [  
  rr:predicate gospl:Music_category_of_Opus ;  
  rr:objectMap [  
    rr:column "category";  
    rr:termType rr:Literal;  
    rr:datatype xsd:string;  
  ];  
];  
rr:predicateObjectMap [  
  rr:predicate gospl:Opus_created_at_Location ;  
  rr:objectMap [  
    rr:column "creation_locatio";  
    rr:datatype xsd:string;  
    rr:termType rr:Literal ; # SHOULD BE BLANKNODE  
  ];  
];  
rr:predicateObjectMap [  
  rr:predicate gospl:Opus_with_subtitle;  
  rr:objectMap [  
    rr:column "subtitle";  
    rr:datatype xsd:string;  
    rr:termType rr:Literal;  
  ];  
];  
rr:predicateObjectMap [  
  rr:predicate gospl:Opus_created_on_Date ;  
  rr:objectMap [  
    rr:column "creation_date";  
  ];  
]
Listing D.1: BOZAR mapping
Appendix E
Agenda.be mappings

```xml
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix gospl: <http://starpc18.vub.ac.be:8080/gospl/ontology/2#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix agenda: <http://www.agenda.be/db/>.

<#TM_events>
  a rr:TriplesMap ;
  rr:logicalTable [ rr:xpathQuery """"//Event_Instance"
    ./EventID"
    ./Event_Title_FR"
    ./Event_Title_NL"
    ./Event_Title_EN"
    ./Event_ShortDescription_FR"
    ./Event_ShortDescription_NL"
    ./Event_ShortDescription_EN"
    ./Event_LongDescription_FR"
    ./Event_LongDescription_NL"
    ./Event_LongDescription_EN"
    ./Event_ContactPhone"
    ./Event_BookingPhone"
    ./Event_Fax"
    ./Event_Mail_FR"
    ./Event_Mail_NL"
    ./Event_Price_FR"
    ./Event_Price_NL"
    ./Event_Price_Description_FR"
    ./Event_Price_Description_NL"
    ./Event_Price_Description_EN"
    ./Event_LanguageDescription_FR"
    ./Event_LanguageDescription_NL"
    ./Event_Canceled"
    ./Event_SoldOut"
    ./Event_CategoryDescription_FR"
    ./Event_CategoryDescription_NL"
    ./Event_GroupDescription_FR"
    ./Event_GroupDescription_NL"
    ./Event_Scope_FR"
    ./Event_Scope_NL"
    ./Event_Performer_FR"
  ] .
```
rr:termType rr:Literal;
];
]
rr:predicateObjectMap {
  rr:predicate gospl:Event_with_Description;
  rr:objectMap [
    rr:column "Event_LongDescription_FR";
    rr:language "fr-BE";
    rr:termType rr:Literal;
  ];
];
rr:predicateObjectMap {
  rr:predicate gospl:Event_with_Description;
  rr:objectMap [
    rr:column "Event_LongDescription_NL";
    rr:language "nl-BE";
    rr:termType rr:Literal;
  ];
];
rr:predicateObjectMap {
  rr:predicate gospl:Event_with_Description;
  rr:objectMap [
    rr:column "Event_LongDescription_EN";
    rr:language "en-GB";
    rr:termType rr:Literal;
  ];
];
rr:predicateObjectMap {
  rr:predicate gospl:Event_with_contact_Phone;
  rr:objectMap [
    rr:column "Event_ContactPhone";
    rr:datatype xsd:string;
    rr:termType rr:Literal;
  ];
];
rr:predicateObjectMap {
  rr:predicate gospl:Event_booked_with_Phone;
  rr:objectMap [
    rr:column "Event_BookingPhone";
    rr:datatype xsd:string;
    rr:termType rr:Literal;
  ];
];
rr:predicateObjectMap {
  rr:predicate gospl:Event_with_fax_Phone;
  rr:objectMap [
    rr:column "Event_Fax";
    rr:datatype xsd:string;
    rr:termType rr:Literal;
  ];
];
rr:predicateObjectMap {
  rr:predicate gospl:Event_with_Email_Address;
  rr:objectMap [
    rr:column "Event_Mail_FR";
    ##rr:language "fr-BE";
    ##rr:termType rr:Literal
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Email_Address;
  rr: objectMap [
    rr: column "Event_Mail_NL";
    ##rr: language "nl-BE";
    ##rr: termType rr: Literal;
    rr: termType rr: IRI;
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Event_in_Language;
  rr: objectMap [
    rr: column "Event_LanguageDescription_FR";
    rr: language "fr-BE";
    rr: termType rr: Literal;
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Event_in_Language;
  rr: objectMap [
    rr: column "Event_LanguageDescription_NL";
    rr: language "nl-BE";
    rr: termType rr: Literal;
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Cancel_Status;
  rr: objectMap [
    rr: column "Event_Canceled";
    rr: termType rr: Literal;
    rr: datatype xsd: boolean;
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Sold_Out_Status;
  rr: objectMap [
    rr: column "Event_SoldOut";
    rr: termType rr: Literal;
    rr: datatype xsd: boolean;
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Price;
  rr: objectMap [
    rr: column "Event_Price_FR";
    rr: language "fr-BE";
    rr: termType rr: Literal;
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Price;
  rr: objectMap [
    rr: column "Event_Price_NL";
    rr: language "nl-BE";
    rr: termType rr: Literal;
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Price_Description;
  rr: objectMap [

83
rr:column "Event_Price_Description_FR";
rr:language "fr-BE";
rr:termType rr:Literal;
];

rr:predicateObjectMap [
  rr:predicate gospl:Event_with_Price_Description;
  rr:objectMap [
    rr:column "Event_Price_Description_NL";
    rr:language "nl-BE";
    rr:termType rr:Literal;
  ];
];

rr:predicateObjectMap [
  rr:predicate gospl:Event_with_Price_Description;
  rr:objectMap [
    rr:column "Event_Price_Description_EN";
    rr:language "en-GB";
    rr:termType rr:Literal;
  ];
];

rr:predicateObjectMap [
  rr:predicate gospl:Event_with_Category;
  rr:objectMap [
    rr:column "Event_CategoryDescription_FR";
    rr:termType rr:Literal;
    rr:language "fr-BE";
  ];
];

rr:predicateObjectMap [
  rr:predicate gospl:Event_with_Category;
  rr:objectMap [
    rr:column "Event_CategoryDescription_NL";
    rr:termType rr:Literal;
    rr:language "nl-BE";
  ];
];

rr:predicateObjectMap [
  rr:predicate gospl:Event_with_Group;
  rr:objectMap [
    rr:column "Event_GroupDescription_FR";
    rr:language "fr-BE";
    rr:termType rr:Literal;
  ];
];

rr:predicateObjectMap [
  rr:predicate gospl:Event_with_Group;
  rr:objectMap [
    rr:column "Event_GroupDescription_NL";
    rr:language "nl-BE";
    rr:termType rr:Literal;
  ];
];

rr:predicateObjectMap [
  rr:predicate gospl:Event_with_Scope;
  rr:objectMap [
    rr:column "Event_Scope_FR";
    rr:language "fr-BE";
    rr:termType rr:Literal;
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Scope;
  rr: objectMap [
    rr: column "Event_Scope_NL";
    rr: language "nl-BE";
    rr: termType rr: Literal;
  ];
];

rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Performer;
  rr: objectMap [
    rr: column "Event_Performer_FR";
    rr: language "fr-BE";
    rr: termType rr: Literal;
  ];
];

rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Performer;
  rr: objectMap [
    rr: column "Event_Performer_NL";
    rr: language "nl-BE";
    rr: termType rr: Literal;
  ];
];

rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Performer;
  rr: objectMap [
    rr: column "Event_Performer_EN";
    rr: language "en-GB";
    rr: termType rr: Literal;
  ];
];

rr: predicateObjectMap [
  rr: predicate gospl: DateTimeSpecification_valid_from_Date;
  rr: objectMap [
    rr: column "Event_DateFrom";
    rr: termType rr: Literal;
  ];
];

rr: predicateObjectMap [
  rr: predicate gospl: DateTimeSpecification_valid_until_Date;
  rr: objectMap [
    rr: column "Event_DateTo";
    rr: termType rr: Literal;
  ];
];

rr: predicateObjectMap [
  rr: predicate gospl: Event_with_Partner;
  rr: objectMap [
    rr: termType rr: IRI;
  ];
];

rr: predicateObjectMap [
  rr: predicate gospl: Event_linked_with_Event;
  rr: refObjectMap [
    rr: parentTriplesMap <#TM_events>;
    rr: joinCondition [
Listing E.1: Agenda.be events mapping

```xml
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix gospl: <http://starpc18.vub.ac.be:8080/gospl/ontology/2#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix agenda: <http://wwwagenda.be/db/>.

<# TM_institutions >
a rr:TriplesMap;

rr:logicalTable [ rr:xpathQuery "/Institution_Instance"
./InstitutionID
./Institution_Name_FR
./Institution_Name_NL
./Institution_ContactPhone
./Institution_BookingPhone
./Institution_Fax
./Institution_Mail_FR
./Institution_Mail_NL
./Institution_WebSite_FR
./Institution_WebSite_NL
./Institution_Locate_FR
./Institution_Locate_NL
./Institution_Description_FR
./Institution_Description_NL
./Institution_Description_EN
./Institution_Opening_FR
./Institution_Opening_NL
./Institution_Opening_EN
./Institution_Logo
./Institution_LongTerme
./Institution_Type1_DescriptionFR
./Institution_Type1_DescriptionNL
./Institution_Type1_DescriptionEN
./Institution_Type2_DescriptionFR
./Institution_Type2_DescriptionNL
./Institution_Type2_DescriptionEN
"
];

rr:subjectMap [ rr:class gospl:Institution;
rr:template "http://wwwagenda.be/db/Institution/{InstitutionID}";
```
rr: predicateObjectMap [  
    rr: predicate gospl:Institution_with_Name ;  
    rr: objectMap [  
        rr: column "Institution_Name_FR" ;  
        rr: termType rr: Literal ;  
        rr: language "fr-be"  
    ] ;  
];

rr: predicateObjectMap [  
    rr: predicate gospl:Institution_with_Name ;  
    rr: objectMap [  
        rr: column "Institution_Name_NL" ;  
        rr: termType rr: Literal ;  
        rr: language "nl-BE" ;  
    ] ;  
];

rr: predicateObjectMap [  
    rr: predicate gospl:Institution_with_contact_Phone ;  
    rr: objectMap [  
        rr: column "Institution_ContactPhone" ;  
        rr: termType rr: Literal ;  
    ] ;  
];

rr: predicateObjectMap [  
    rr: predicate gospl:Institution_with_booking_Phone ;  
    rr: objectMap [  
        rr: column "Institution_BookingPhone" ;  
        rr: termType rr: Literal ;  
    ] ;  
];

rr: predicateObjectMap [  
    rr: predicate gospl:Institution_with_fax_Phone ;  
    rr: objectMap [  
        rr: column "Institution_Fax" ;  
        rr: termType rr: Literal ;  
    ] ;  
];

rr: predicateObjectMap [  
    rr: predicate gospl:Institution_with_Email_Address ;  
    rr: objectMap [  
        rr: column "Institution_Mail_FR" ;  
        rr: termType rr: IRI ;  
    ] ;  
];

rr: predicateObjectMap [  
    rr: predicate gospl:Institution_with_Email_Address ;  
    rr: objectMap [  
        rr: column "Institution_Mail_NL" ;  
        rr: termType rr: IRI ;  
    ] ;  
];

rr: predicateObjectMap [  
    rr: predicate gospl:Institution_with_Website ;  
    rr: objectMap [  
        rr: column "Institution_WebSite_FR" ;  
        rr: termType rr: IRI ;  
    ] ;  
];
rr: predicateObjectMap [
  rr: predicate gospl: Institution_with_Website;
  rr: objectMap [
    rr: column "Institution_WebSite_NL";
    rr: termType rr: IRI;
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Institution_with_Location;
  rr: objectMap [
    rr: column "Institution_Locate_FR";
    rr: termType rr: Literal;
    rr: language "fr-BE";
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Institution_with_Location;
  rr: objectMap [
    rr: column "Institution_Locate_NL";
    rr: termType rr: Literal;
    rr: language "nl-BE";
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Institution_with_Description;
  rr: objectMap [
    rr: column "Institution_Description_FR";
    rr: termType rr: Literal;
    rr: language "fr-BE";
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Institution_with_Description;
  rr: objectMap [
    rr: column "Institution_Description_NL";
    rr: termType rr: Literal;
    rr: language "nl-BE";
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Institution_with_Description;
  rr: objectMap [
    rr: column "Institution_Description_EN";
    rr: termType rr: Literal;
    rr: language "en-GB";
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Institution_with_Opening;
  rr: objectMap [
    rr: column "Institution_Opening_FR";
    rr: termType rr: Literal;
    rr: language "fr-BE";
  ];
];
rr: predicateObjectMap [
  rr: predicate gospl: Institution_with_Opening;
  rr: objectMap [
    rr: column "Institution_Opening_NL";
    rr: termType rr: Literal;
  ];
];
rr:language "nl-BE";
];
rr:predicateObjectMap [  
  rr:predicate gospl:Institution_with_Opening ;  
  rr:objectMap [  
    rr:column "Institution_Opening_EN" ;  
    rr:termType rr:Literal ;  
    rr:language "en-GB" ;  
  ] ;
];
rr:predicateObjectMap [  
  rr:predicate gospl:Institution_with_Logo ;  
  rr:objectMap [  
    rr:column "Institution_Logo" ;  
    rr:termType rr:Literal ;  
  ] ;
];
rr:predicateObjectMap [  
  rr:predicate gospl:Institution_has_Longterme_Status ;  
  rr:objectMap [  
    rr:column "Institution_LongTerme" ;  
    rr:termType rr:Literal ;  
    rr:datatype xsd:boolean ;  
  ] ;
];
rr:predicateObjectMap [  
  rr:predicate gospl:Institution_with_Type ;  
  rr:objectMap [  
    rr:column "Institution_Type1_DescriptionFR" ;  
    rr:termType rr:Literal ;  
    rr:language "fr-BE" ;  
  ] ;
];
rr:predicateObjectMap [  
  rr:predicate gospl:Institution_with_Type ;  
  rr:objectMap [  
    rr:column "Institution_Type1_DescriptionNL" ;  
    rr:termType rr:Literal ;  
    rr:language "nl-BE" ;  
  ] ;
];
rr:predicateObjectMap [  
  rr:predicate gospl:Institution_with_Type ;  
  rr:objectMap [  
    rr:column "Institution_Type1_DescriptionEN" ;  
    rr:termType rr:Literal ;  
    rr:language "en-GB" ;  
  ] ;
];
rr:predicateObjectMap [  
  rr:predicate gospl:Institution_with_Type ;  
  rr:objectMap [  
    rr:column "Institution_Type2_DescriptionFR" ;  
    rr:termType rr:Literal ;  
    rr:language "fr-BE" ;  
  ] ;
];
rr:predicateObjectMap [  
  rr:predicate gospl:Institution_with_Type ;  
  rr:objectMap [  
    rr:column "Institution_Type2_DescriptionNL" ;  
    rr:termType rr:Literal ;  
    rr:language "nl-BE" ;  
  ] ;
];
rr:predicateObjectMap [  
  rr:predicate gospl:Institution_with_Type ;  
  rr:objectMap [  
    rr:column "Institution_Type2_DescriptionEN" ;  
    rr:termType rr:Literal ;  
    rr:language "en-GB" ;  
  ] ;
];
Listing E.2: Agenda.be activities mapping
Bibliography

http://d2rq.org/d2rq-language.

http://docs.openlinksw.com/virtuoso/.

[3] SPARQL Query Language for RDF. 
http://www.w3.org/TR/rdf-sparql-query/.


http://www.w3.org/TR/2011/WD-r2rml-20110920/.

http://www.w3.org/TR/r2rml/.

http://www.springerlink.com/content/rm32474088w54378/fulltext.pdf.


[http://www.w3.org/DesignIssues/LinkedData.html](http://www.w3.org/DesignIssues/LinkedData.html)

[http://www.w3.org/TR/rdb-direct-mapping/](http://www.w3.org/TR/rdb-direct-mapping/)

