Publishing Relational Data as Linked Data

Teresa Isabel Almeida da Costa
Master’s Degree in Network and Information Systems Engineering
Department of Computer Science
2013

Supervisor
José Paulo Leal, Assistant Professor,
Faculty of Sciences of the University of Porto
Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/_________
Abstract

Information on the web is mostly targeted to humans. Any average Internet user is able to purposefully browse web pages and understand their content, taking advantage of a graphical formatting designed to enhance human perception. However the same thing does not apply to software and is very difficult to build tools capable of extracting and relating information retrieved from different web sources.

Fortunately, most data available on the web is stored in databases. If we were able to publish all this data using common vocabularies and identifiers it would allow applications to access and use this raw data, creating a wide net of interlinked data, independent of its origins and internal organization.

This dissertation describes the development of the DaPress system, which transforms relational data into semantically structured data and publishes it on the web. This tool implements algorithms to map the structure and content of relational databases. This process is driven by an XML configuration file and stores the produced RDF in triplestores – specialized databases. It also allows access to the data through a SPARQL web service.

The dissertation starts with a survey of the technologies, languages and specifications used in the scope of Linked Data, such as RDF, RDF Schema and OWL. This survey covered also related systems used to publishing Linked Data on the Web.

The validation of DaPress was made using the Authenticus database. Authenticus is a system that automatically assigns scientific publications to their authors and institutions. Smaller databases from other domains were also mapped, to complement this validation.
Resumo

A informação na web está direcionada para humanos. Qualquer utilizador da Internet consegue navegar pelas páginas web e compreender o seu conteúdo, tirando partido da sua formatação gráfica desenhada para aumentar a percepção humana. Contudo, isto não se aplica ao software e é muito difícil contruir ferramentas capazes de extrair e relacionar informação que tenha diferentes origens.

Felizmente, a maior parte dos dados disponíveis na web estão armazenados em bases de dados. Se fossemos capazes de publicar toda essa informação usando vocabulários e identificadores comuns isso permitiria que as aplicações acedessem e usassem os dados em bruto, criando uma extensa rede de dados interligados, independentemente da sua origem e da sua organização interna.

Esta dissertação descreve o desenvolvimento do sistema DaPress, que transforma dados relacionais em dados semanticamente estruturados e os publica na web. Esta ferramenta implementa algoritmos que mapeiam a estrutura e o conteúdo de bases de dados relacionais. Este processo é conduzido por um ficheiro de configuração em XML e armazena os triplos gerados em triplestores – bases de dados especializadas. Também permite o acesso aos dados através de um serviço web de SPARQL.

Esta dissertação começa com uma pesquisa de tecnologias, especificações e linguagens usadas no âmbito de Linked Data, nomeadamente RDF, RDF Schema e OWL. Esta pesquisa também abrangiu sistemas usados para publicar Linked Data na web.

A validação do DaPress foi feita usando a base de dados do Authenticus. O Authenticus é um sistema que atribui de forma automática publicações científicas aos seus autores e instituições. Outras bases de dados de outros domínios foram igualmente utilizadas, para complementar esta validação.
Acknowledgments

I would like to acknowledge several people that in some way contributed to this thesis:

To my supervisor, Prof. José Paulo Leal, thank you very much for your help, guidance and support during these months. I have learned so much with you and I am grateful for this great opportunity.

To all my friends, thank you always for being there. A special thanks for my friend, Filipe, for all the coffees after lunch, talks, laughs and support that have helped me in so many ways.

To Pedro, my better half, soul mate and best friend, thank you for being always by my side. Without your love, care and patience it would be much more difficult.

To my brother, Tiago, thank you. You are an inspiration to me and these last years were an amazing ride.

Finalmente, aos meus pais. Não tenho palavras para vos agradecer tudo o que fazem por mim. O vosso apoio e amor incondicional fazem-me acreditar que tudo é possível. Muito obrigada por me apoiarem na vida que escolhi.
Contents

Abstract v
Resumo vii
List of Tables xiii
List of Figures xiv
Listings xv

1 Introduction 2
1.1 Motivation 2
1.2 Goals and Approach 4
1.3 Overview 5

2 Semantic Web 7
2.1 Resource Description Framework 8
2.1.1 Data Model 9
2.1.2 Persistence 14
2.1.3 Interrogation Language 14
2.1.4 Serialization 16
2.2 Ontologies 19
2.2.1 Resource Description Framework Schema 20
2.2.2 Web Ontology Language: OWL 23
2.2.3 Comparing RDF Schema and OWL 28
2.2.4 Common Vocabularies 29

3 Web of Data 35
List of Tables

2.1 Textual representation of the sentence. 12
2.2 FOAF classes and Properties. 32
2.3 Dublin Core elements. 33

3.1 A table with the movies playing this week. 37
3.2 A table with the movies playing this week with hours. 37
3.3 Example of table listing temperatures over a time period. 40
3.4 Example of schema information related to Figure 3.3. 40

4.1 Original database. 60
4.2 Database specifications and size related results. 64
4.3 Time related results. 64
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Semantic Web Architecture.</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>A Venn diagram representing the relationship between URIs and URLs.</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Representation of two related persons using blank nodes.</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Simple example of graph-based triples representation</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>RDF and RDFS example</td>
<td>22</td>
</tr>
<tr>
<td>2.6</td>
<td>FOAF classes diagram.</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>Simple movie list schema.</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>Movie list relational data.</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Linking Open Data cloud diagram</td>
<td>42</td>
</tr>
<tr>
<td>3.4</td>
<td>Apache Jena Framework architecture</td>
<td>44</td>
</tr>
<tr>
<td>3.5</td>
<td>Sesame Architecture.</td>
<td>45</td>
</tr>
<tr>
<td>3.6</td>
<td>D2RQ Architecture.</td>
<td>47</td>
</tr>
<tr>
<td>3.7</td>
<td>OpenLink Virtuoso architecture</td>
<td>48</td>
</tr>
<tr>
<td>4.1</td>
<td>DaPress Architecture.</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Screenshot of the DaPress Access Point.</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>XML schema of DaPress configuration files</td>
<td>59</td>
</tr>
<tr>
<td>4.4</td>
<td>Example of the algorithms application.</td>
<td>62</td>
</tr>
<tr>
<td>C.1</td>
<td>Authenticus UML Diagram.</td>
<td>73</td>
</tr>
<tr>
<td>C.2</td>
<td>DBLP UML Diagram.</td>
<td>74</td>
</tr>
</tbody>
</table>
List of Listings

2.1 SPARQL query example. ........................................ 15
2.2 An example of RDF/XML serialization. ....................... 17
2.3 N-Triple serialization example. ............................... 18
2.4 N3 serialization example. ...................................... 18
2.5 An example of the import section. ............................ 24
2.6 An example of property restriction. ........................... 26
2.7 Describing a person with FOAF. ............................... 31
B.1 XML Configuration Document Example ........................ 72
Chapter 1

Introduction

1.1 Motivation

The World Wide Web has deeply changed the world. Since its beginning it connected people from different places and promoted a new era of sharing knowledge and opportunities. Nevertheless, most users could only view web pages and not contribute to their content. The information was static and could only be updated by webmasters. User-generated content was largely unavailable.

The second wave of the Web – usually called Web 2.0 – allowed the interaction and collaboration among users, creating dynamic web pages with user-generated content. This was closer to the original vision of Tim Berners-Lee of a “collaborative medium, a place where we could all meet, read and write”[36, 31].

Nowadays it is easy to produce HTML formatted content and publish it in an HTTP server. Virtually, anyone, anywhere on the planet, can access it using a web browser and benefit from its content. Anyone but not anything. The information is produced and formatted for humans. It is simple, for a person, to understand web content and navigate through hyperlinks with a meaningful purpose. Humans are very flexible data processors and are able to extract information from a web page whether it is arranged as a table, an outline or a multipage narrative.

However, it is very hard to build a software agent that gathers information from the web for a fairly simple task, such as setting an appointment with a doctor or planning a
business trip. It is still a challenge after more than a decade of research. If the diversity of information on the Web were available on common and machine-understandable formats, then tools could access and use it. In this vision, data from different sources are interlinked and it is possible to create new knowledge derived from the combination of independent data sources. This is the vision of the semantic web.

The goal of the semantic web is to open the vast amount of data available on the web to software processing. The semantic technologies allow the user to focus on the behavior of the application instead of the data processing. The first attempt was to markup with semantic annotations the content already available on web pages. The use of XML languages and separation of content from formatting was expected to contribute to that goal. However, the forces that shape the evolution of the web clearly favor graphical user interaction over semantic content. Hence, nowadays it is harder to provide semantic annotations to web applications and web services than it was to last century hand-made web pages.

Fortunately, most web content is actually generated from databases. Thus, instead of extracting information from web pages, it is more effective to collect it directly from raw data sources. The linked data initiative promotes best practises for publishing data that supports web content. This data should be published in open formats so that it can be read and processed by any software. Moreover data from different sources should be interlinked to create a global web content.

Navigation on the world wide web relies on content being linked using Uniform Resource Locations (URLs). Linked data follows a similar approach to enable software agents to navigate through data available from different sources. If URIs are used as identifiers the content of a database may refer the content of another.

Interoperability has been a concern in databases for a long time. A typical relational database management system imports and exports data in open formats, such as XML or comma-separated values (CSV), and relational databases themselves are based on open standards, such as the Structured Query Language (SQL).

Unfortunately these open standard formats are not enough to build linked data.

1. The structure of a relational database is rigid. The software that processes a relational data is designed and implemented for a particular database schema
and needs to be updated to reflect changes in that schema. A program to process
a generic relational database, independently of its schema would be too complex
to implement.

2. The semantics of data stored in relational databases is not explicit. An application
that processes relational data relies on an implicit knowledge of the meaning of
the data, and linking related data from different sources is a difficult task. To a
human it may be obvious that the tables named “teacher” and “lecturer” from two
different academic databases contains similar data. But that kind of reasoning is
extremely difficult to automatise.

3. A typical relational database contains both data that should be published mixed
with sensitive or irrelevant data that should not be published. Also, publishable
data may need to be preprocessed to normalize either its content or its structure.

1.2 Goals and Approach

The main objective of this work is the development of a simple and flexible approach
to publish the content of relational databases as linked data on the Web. The system
implementing this approach must provide a SPARQL access point over a web service,
where the content of a relational database, or part of it, is retrieved in RDF format. The
development of this approach was guided by a number of design goals, from which the
following are the most relevant.

- The approach must be applicable to any existing relational databases.
- The access point should be independent from the original relational database.
- The published data should cover both the schema and the content of database.
- The approach should provide control over how and what data is published.

To ensure that DaPress works with any relational database it was necessary to design
a configuration file, independent of the application itself, where the information needed
to achieve all the application features is described. It contains information about the
relational database to map such as the type and the credentials. This configuration
also contains the tables and fields that will be mapped and with which name, offering
full control over the data published.

Using the configuration file, two algorithm were implemented to execute the mapping of both the content and the schema of the relational database. Both algorithms use information from the configuration file and from the relational database. The main difference is that mapping the content resources are related to objects through properties while mapping the schema classes are related to values through predefined RDF, RDFS and OWL properties.

The SPARQL interrogation point was also based on the configuration file that contains the definition of the triplestore in use. Using the same triplestore is also possible to query different graphs.

In order to build the configuration file, a XML Schema was designed providing a formal description about the content of the file. The framework chosen for the development of the DaPress was the Apache Jena Framework. This is a well known and documented framework that provides all the APIs required to implement the DaPress features. Jena also provides several types of triplestores. Since this work is related to relational databases the triplestore chosen was the SDB.

1.3 Overview

The body of this dissertation is organized in four chapters. The next two chapters provide the relevant background. The following chapter presents the proposed approach for publishing relational data as linked data. Several databases were published using this approach and the results are also discussed in this chapter. The last chapter draws conclusions on the presented work and highlights possible future work. The remainder of this section introduces each of these chapters.

Chapter 2 introduces the Semantic Web and its most prominent languages; RDF, RDF Schema and OWL are describe in detail and in relation to each other.

Chapter 3 provides an introduction to Web of Data and Linked Data. It describes how data is organized, comparing the data models of spreadsheets and relational databases, with those used for semantic data. The final section presents frameworks that are used for the development of semantic web applications and existing systems
for publishing Linked Data.

Chapter 4 presents the proposed approach and DaPress, the tool implementing it. It introduces the strategy of mapping relational data into linked data and explains in detail the algorithms for mapping both content and schemata respectively into RDF and OWL. It presents the architecture of DaPress and describes in detail each module, with emphasis on the configuration documents that control the mapping algorithms. It also presents the web service that provides a linked data access point to the database. Finally, for validation of this approach, examples of existing relational databases mapped linked data with DaPress are also reported.

The last chapter draws some conclusions about the proposed work for building a tool to map relational data into linked data. It points out the challenges lying ahead and identifies future work in this area of research.

The dissertation also includes the appendix with acronyms, a configuration file and UML diagrams. The last part contains the bibliographic references.
Chapter 2

Semantic Web

The world wide web is based on computers but targeted to people. The web pages use natural language, images and page layouts to present information in a way that it is easy for a human to understand.

The Semantic Web [11, 2] is a collaborative movement led by the World Wide Web Consortium. The main goal of Semantic Web is to open the information on the web to software processing. The idea is to add metadata – knowledge about how the data can be used – to the web pages making the world wide web machine readable.

Figure 2.1: Semantic Web Architecture.

source: www.w3c.org

The Semantic Web provides a common framework that allows data to be shared and
reused across applications. Its components are deployed in the layers of Web technologies and specifications as represented in the Figure 2.1. There are five main components of the Semantic Web:

URI - Uniform Resource Identifier: is a format for web identifiers that is widely used on the World Wide Web. The Semantic Web uses URIs to represent most kinds of data.

RDF - Resource Description Framework: is used by Semantic Web to describe data uniformly, allowing it to be shared. It is a general metadata format used to represent information about Internet resources. It extends the expressive capability of Web augmenting human-readable web pages with machine-processable information.

RDF Schema: is a language used by the Semantic Web to describe the data properties used in RDF. It provides mechanisms for describing resources and relationships between these resources. The RDFS vocabulary descriptions are also RDF.

Ontologies are used to represent the structure of knowledge domain. The Semantic Web uses OWL, Web Ontology Language. Applications need that language in order to process data rather than just display it. OWL adds the possibility of reasoning to data by identifying and describing relationships between data items. Ontologies are defined independently from the actual data and reflect a common understanding of the semantics of the domain. It provides definitions of classes, relations, functions, constraints and other objects.

Logic - Inference is useful to derive new data from data. A common example is the property transitivity. If an element has a type $A$ and the type $A$ is a subtype of type $B$ then the element has also the type $B$.

2.1 Resource Description Framework

The Resource Description Framework (RDF)[29, 2, 38, 34, 30] is a framework for representing any kind of information available in the Web. According to the W3C, RDF is a language designed to support the Semantic Web, providing structures that can be
used for interoperable XML data exchange. RDF provides a standard way of encoding and exchanging data with other people, and specially with machines, without loss of meaning.

RDF's purpose is straightforward: it provides a means of recording data in a machine-understandable format that allows more efficient and sophisticated data interchange, searching, cataloging, navigation and classification. RDF is based on the concept "Web Resource". It can be used to represent information about things that can be identified on the Web, even if it cannot be directly retrieved from the Web.

This section introduces the RDF standards. The RDF data model (Subsection 2.1.1), how the data graph can be persistently stored (Subsection 2.1.2), the several ways of serialize the information stored in a graph (Subsection 2.1.4) and the specific language to query the data model (Subsection 2.1.3) are described in more detail.

2.1.1 Data Model

The RDF data model provides an abstract, conceptual framework for defining and using metadata. It has a graph-based data model that eases the processing and manipulation by applications.

The basic element in RDF is a statement. It is a simple sentence with three parts - subject, predicate and object - expressing a relationship between things. The subject is a resource, the thing to describe and it is identified by an URI on a blank node. The predicate specifies an aspect, characteristic, attribute or relation used to describe the subject. It is also identified by an URI. A specific resource, together with a named property, needs an object in order to construct a statement. The object can either be a resource, a blank node or an atomic value, called literal. Being composed by three parts, an RDF statement is also known as triple.

A directed labeled multi-graph describes the RDF data model and it is very easy to read. A collection of triples forms a graph; the nodes are given by the resources and objects of triples and the arcs that connect them are the properties, given by predicates. They form a pattern of node - arc - node. The nodes come in three varieties: URI, blank nodes and literals.
The **URIs** are used as strong keys. They assign unique identifiers to each of the nodes so that can be referred consistently across all the triples that describe their relationships. They always represent the same thing, regardless of the context where they are found. They provide a common syntax for naming a resource regardless of the protocol used to access the resource. URIs are related to URLs in that a URL is a specific instance of an URI schema based on a known protocol. The Figure 2.2 shows how URIs, URLs and URNs are related.

**Figure 2.2**: A Venn diagram representing the relationship between URIs and URLs.

The fundamental difference between an URL and an URN is that URL specifies where an identified resource is available and the mechanism for retrieving it. A URN is an URI that uses the URN scheme and does not imply the availability of the identified resource. Both URNs (names) and URLs (locators) are URIs, and a particular URI may be both a name and a locator at the same time. Currently, there are a generalization of the URI – IRI – that may contain characters from the Universal Character Set enabling non-Latin alphabet users to use it. It is common in RDF to shorten URIs by assigning a namespace to the base URI and writing only the distinctive part of the identifier. The W3C has defined a number of standard namespaces for use with Web technologies.

**rdf**: indicates identifiers used in RDF used to identify types and properties in RDF. The global URI is [http://www.w3.org/1999/02/22-rdf-syntax-ns#](http://www.w3.org/1999/02/22-rdf-syntax-ns#).

**rdfs**: The global URI is [http://www.w3.org/2000/01/rdf-schema#](http://www.w3.org/2000/01/rdf-schema#) and indicates iden-
tifiers used for the RDF Schema language that are explained in more detail in the Subsection 2.2.1.

**owl**: The global URI is `http://www.w3.org/2002/07/owl#` and indicates identifiers used for the Web Ontology Language that are explained in more detail in the Subsection 2.2.2.

The **blank nodes** are graph nodes that also represent a subject or object for which is possible to make assertions but there is no way to address with a proper URI. Most of RDF APIs handle this type of nodes using an internal ID for the node that is only valid in the local graph and cannot be used as a strong key between graphs.

Figure 2.3 depicts a graph with blank nodes referring to persons that are not identified by a strong URI. Still, using blank nodes one is able to use them meaningfully in several interrelated statements.

![Figure 2.3](http://example.org/bday)

The **literals** consist of three parts - a character string, an optional language tag and a data type. They represent only objects, never subjects or predicates. RDF supports both user defined and XML Schema types, including basic data types, such as Integer, Boolean, Time or Date. For typing complex concepts, such as resources and properties one must resort to RDF schema, as explained in the Subsection 2.2.1.

Lets consider another simple example to show two different ways of represent a statement.
CHAPTER 2. SEMANTIC WEB

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>..../Person/QT</td>
<td>../name</td>
<td>&quot;Quentin Tarantino&quot;</td>
</tr>
<tr>
<td>..../Movie/PF</td>
<td>../title</td>
<td>&quot;Pulp Fiction&quot;</td>
</tr>
<tr>
<td>..../Person/QT</td>
<td>../directorOf</td>
<td>../Movie/PF</td>
</tr>
</tbody>
</table>

Table 2.1: Textual representation of the sentence.

Quentin Tarantino is the director of Pulp Fiction.

The following table shows the triples extracted from the previous sentence. For sake of terseness, ellipsis replaces a common URL prefix. Note that the concepts “Quentin Tarantino” and “Pulp Fiction” where replaced by URI as was the “is director of” property. By cultural context, it is known that Quentin Tarantino is a person’s name and Pulp Fiction a movie title. Using that information the two other statements assign a textual representation to both subject and object of the previous sentence.

The Figure 2.4 is a graph and equivalent representation of the same three triples. It is a directed graph, with labeled nodes and arcs. The arcs are directed from the resource (the subject) to the value (the object). This kind of graph is known as Semantic Net.

Figure 2.4: Simple example of graph-based triples representation
Regardless of the manner which and RDF triple is written they describe the same model there are three immutable facts about each:

1. is made up of subject, predicate and object;

2. is a complete and unique fact;

3. can be linked to other RDF triples, but it still retains its own unique meaning, regardless of the complexity of the model in which it is included.

RDF also provides additional capabilities for representing groups of resources and statements. To describe a group of things, RDF offers several predefined types and properties. A container is a resource that contains things, called members. These members may be resources or literals. There are three types of containers: the rdf:Bag, the rdf:Seq and the rdf:Alt. To represent a group of members, where duplicate members could exist, and without significant order of the members, the Bag (rdf:Bag) is used. If the order of the members is relevant then it is the Sequence (rdf:Seq) that defines the set. The element rdf:Alt, as in Alternative, is used for a group of resources or literals that are alternatives, usually to the value of a property. To describe a resource as being one of these types of containers, the value of the rdf:type should be rdf:Bag, rdf:Seq or rdf:Alt.

The containers are not enough to specify a closed group of elements. To perform this task, RDF offers collections. An RDF collection is a group of things represented as a structured list in the graph. The list is constructed using the predefined type rdf:List, the properties rdf:first and rdf:rest and the resource rdf:nil. Each member of the collection as a rdf:first property whose subject is a resource that represents the list. The list resource is linked to the rest of the list by the rdf:rest property. The end of the list is given by the resource rdf:nil.

Sometimes it is necessary to make statements about relationships. The process of making a subject-predicate-object statement into a subject is called reification. The RDF reification consists of the element rdf:Statement and the properties rdf:subject, rdf:predicate and rdf:object. The typical use of RDF reification vocabulary involves describing a statement using four statements in this pattern (the “reification quad”).
2.1.2 Persistence

Data in the RDF data model can be persistently stored in a special type of database called triplestore. Triplestores are Database Management Systems (DBMS) for data modeled using RDF. While Relational Database Management Systems (RDBMS) store data in relations (or tables) and are queried using SQL, triplestores store RDF triples and are queried using SPARQL.

Triplestores can be classified in three categories: native triplestores, RDBMS-backed triplestores and NoSQL triplestores. Native triplestores are implemented from scratch and exploit the RDF data model to efficiently store and retrieve data. This category includes Jena TDB, Sesame and 4Store. RDBMS-backed triplestores are built by adding an RDF specific layer to an existing RDBMS. An example of this type of triplestores is Jena SDB or Virtuoso. NoSQL triplestores are more recent and still under investigations. These triplestores are built over NoSQL databases. An example is CumulusRDF that is built on top of Cassandra.

Triplestores provide schema flexibility and standardization. They are prepared to handle different data types and schemata that evolve over the time. Since they are based on standards the process of moving data between triplestores is very simple.

However, there are downsides on using triplestores. SQL databases are much more mature and have more features than an RDF database. Transactions are more crude and the cost per unit information stored in RDF is much higher than in a SQL database. This is very significant when the amount of data is huge.

2.1.3 Interrogation Language

Just as SQL provides a standard query language across the relational database systems, SPARQL (Simple Protocol and RDF Query Language) [21, 35] provides a declarative interface for interacting with RDF graphs. SPARQL is both a standard query language and a data access protocol. This language consists of a triple of patterns, conjunctions (logical “and”) and disjunctions (logical “or”). Variables in the triple pattern are identified by character strings starting with a question mark (?).

Using the RDF described in the Figure 2.4 is possible to write a simple SPARQL query.
PREFIX ex: <http://example.org>
SELECT ?title
WHERE {
?y ex:name 'Quentin Tarantino';
ex:directorOf ?z.
}

Listing 2.1: SPARQL query example.

This query interrogates the RDF model asking which movie titles Quentin Tarantino has directed. The query result should be:

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Pulp Fiction&quot;</td>
</tr>
</tbody>
</table>

SPARQL provides four types of queries: SELECT, CONSTRUCT, ASK and DESCRIBE. All these forms attempt to find solutions to a graph pattern. A SPARQL query has the following type:

PREFIX (optional)
SELECT | CONSTRUCT | DESCRIBE | ASK ...
FROM ...
WHERE ...
FILTER (optional) OPTIONAL (optional)

The PREFIX clause is optional. These declarations assign shorthand identifiers to URI namespaces that can be used through the query. The SELECT clause identifies the values to be returned. It returns all, or a subset of, the variables bound in a query pattern match. The FROM clause identifies the data sources to query. The WHERE clause identifies the triple or graph pattern to be matched against the triples or graphs of RDF.

The FILTER and OPTIONAL constraints are both optional. While the FILTER is used to add constraints to the graph pattern, OPTIONAL is used to make the matching of a part of the pattern optional being very useful for handling missing data. SPARQL FILTER operators also allow to set up conditions on the qualities of a bound variable value.

In addition to the query form SELECT, SPARQL provides other ways to interrogate an RDF. When it is necessary to construct a new graph with the query results SPARQL provides the CONSTRUCT clause. The graph is formed by each query solution sequence,
substituting for the variables in the graph template and combining the triples into a single RDF graph by union. Mostly for testing purposes, a simple \texttt{ASK} tests whether a pattern can be found in a graph returning a boolean result indicating if the pattern is in the graph. The \texttt{DESCRIBE} clause is also available and provides information about the resources returned in a solution.

A SPARQL query is executed against a dataset. An RDF dataset comprises a default graph which does not have a name and zero or more named graphs that are identified by an IRI. With SPARQL it is possible to interrogate the multiple graphs stored, matching different parts of the query patterns against different graphs. Two different keywords are used to specify the queried dataset. The \texttt{FROM} keyword contains an IRI that indicates a graph to be used to form the default graph. If the query has more than one \texttt{FROM} clause, the default graph is based on the merge of the multiple graphs. The keyword \texttt{FROM NAMED} indicates the named graphs in the dataset to query. It is combined with the \texttt{FROM} keyword that, if not specified, an empty graph is included as default.

SPARQL also provide means to combine different patterns so that one of several alternative graph patterns may match. If more than one alternative matches, all the pattern solutions are found. This alternatives are specified using the \texttt{UNION} keyword.

With SPARQL 1.1 \cite{26}, graph update and management became available. The graph update includes triple management, adding or removing triples. The graph management implies create new graphs or delete them, drop data from a graph, move and copy data among graphs or add a graph to another graph.

\subsection{2.1.4 Serialization}

As said before, RDF is not a data format but a data model for describing resources in the form of subject, predicate, object triples. In order to publish an RDF graph on the Web it is necessary to serialize the data in the RDF graph, using its syntax. The serialization formats can be both XML and non-XML based.

\texttt{RDF/XML} \cite{30} is the oldest RDF serialization format. It is part of the original RDF specification in 1999. This format was widely used to publish Linked Data on the Web. However, the syntax is difficult for humans to read and write. The following code is a
simple example of an RDF/XML serialization.

```xml
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ex="http://example.org">
  <rdf:Description rdf:about="http://example.org/Person/QT">
    <ex:name>Quentin Tarantino</ex:name>
    <ex:bday>27/03/1963</ex:bday>
    <ex:directorOf rdf:resource="http://example.org/Movie/PF">
    </ex:directorOf>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Movie/PF">
    <ex:title>Pulp Fiction</ex:title>
  </rdf:Description>
</rdf:RDF>
```

Listing 2.2: An example of RDF/XML serialization.

The RDF/XML is built up from a series of smaller descriptions each of which traces a path through an RDF graph. The path is described in terms of subjects (nodes) and links (predicates) connecting the nodes. If there are several paths described in the document, all the descriptions must be children of a single `rdf:RDF` element, that is, an `RDF` element from the RDF namespace. The subject of each triple is identified in the `rdf:about` attribute of the `rdf:Description` element. Literal objects are expressed as plain text (or in XML terms, as PCDATA) between the start- and end-tags. If the object value is a resource it is identified using `rdf:resource`.

As in other XML languages, the top-level element is frequently used to define XML namespaces used through the document. RDF/XML also includes a number of rules for determining the fully qualified URI of a resource mentioned in this type of document. This feature can be both an advantage and a limitation to the use of the RDF/XML since it does not allow the serialization of a non-NCName char. For example, a property with the URI `http://example.org/prop(erty)` cannot be serialized.

The RDF/XML serialization is rather complex and its documents are difficult to process. The following types of serialization – namely N-Triples, Notation 3 and Turtle – are text based but non-XML formats are easy to produce, process and understand.

The **N-Triple** [5] notation is the simplest form of RDF serialization and corresponds directly to raw RDF triples, which make it rather verbose. The simplicity of this format
makes it very useful when hand-crafting datasets for testing and debugging purposes. An example of this notation follows:

```xml
<http://example.org/Person/QT> <http://example.org/name> "Quentin Tarantino".
<http://example.org/Person/QT> <http://example.org/bday> "27/03/1963".
<http://example.org/Person/QT> <http://example.org/directorOf> <http://example.org/Movie/PF>.
<http://example.org/Movie/PF> <http://example.org/title> "Pulp Fiction".
```

Listing 2.3: N-Triple serialization example.

In this type of serialization there are no prefixes and the fully qualified URI is included in each statement. Each line of the document represents a single statement containing a subject, predicate and object followed by a dot (\(\)). Except for blank nodes and literals, subjects, predicates and objects are expressed as absolute URIs, enclosed in angle brackets. Subjects and objects representing anonymous nodes are represented as \_\_name, where \_\_name is an alphanumeric string that starts with a letter. Object literals are double-quoted strings. String literals can have specified its language by using lang, where lang is the language code. Literals can also provide information about their datatype when followed by \^^ type, where type is a XSD (XML Schema Definition) datatype.

The Notation 3 [9] format, also known as N3, is another notation used frequently. This serialization format is a personal project by Tim Berners-Lee that he describes as basically equivalent to RDF in its XML syntax, but easier to scribble when getting started[7]. N3 combines the simplicity of N-Triples with RDF/XML ability to abbreviate long URIs with prefixes and it also introduces new features. The following code shows how a simple model is serialized in this format.

```xml
@prefix ex: <http://example.org> .
<http://example.org/Person/QT> ex:name "Quentin Tarantino";
ex:bday "27/03/1963";
ex:directorOf <http://example.org/Movie/PF>.
<http://example.org/Movie/PF> ex:title "Pulp Fiction".
```

Listing 2.4: N3 serialization example.

This serialization is very similar to N-Triples, using the form subject, predicate, object and dot. However, it allows the use of formatting and the use of namespace prefixes.
to make statements shorter. N3 also offers a shortcut for describing multiple facts about the same subject, reducing the repetition, by using a semicolon (;) after the first statement. The final object end with a period to show that the sentence is complete. A comma (,) is also used and it means that the subject and predicated are shared with several objects. With N3, it is possible to express a group of statements that share a common anonymous subject without having to specify an internal name for the blank node, having the sentence enclosed in squared brackets. N3 also introduces other interesting features, such as the ability to refer to a graph of triples as a resource and inference rules.

The **Turtle** [8] stands for Terse RDF Triple Language and it is a subset of the Notation 3 format. It is a very popular and simple serialization format. While Notation 3 has an expressive power that goes much beyond RDF, Turtle is a subset of the minimal features of N3 that only serializes valid RDF graphs.

### 2.2 Ontologies

Ontologies are used to formally represent knowledge as a set of concepts from a domain and the relationships between these concepts. By definition, an ontology is a *formal specification of a shared conceptualization* [27]. It provides a shared vocabulary which can be used to model a domain, the type of objects and concepts that exist, and their properties and relations. Ontologies are described in a way that allows abstraction from the data structures and implementation strategies. For this reason, ontologies are said to be at the “semantic” level while the data model is in at the “physical” level. Being independent from the lower level data models, ontologies are used for heterogeneous databases integration, enabling interoperability between different systems.

In the Semantic Web, there is no clear division between ontologies and vocabularies. However, the term “ontology” is used for more complex and formal collection of terms while the term “vocabulary” is used when it is unnecessary a strict formalism. The role of vocabularies in this field is to help data integration between different datasets or when extra knowledge may lead to discover new relationships. Another use for ontologies is the knowledge organization in many different scopes, such as museums and libraries or other communities that manages large collections of data. By using standard and
common vocabularies is possible to combine knowledge from different sources and that leverages the power of linked data.

To fulfil these different needs, W3C offers several techniques to describe and define vocabularies in a standard format, such as Resource Description Framework Schema described in the Subsection 2.2.1 and Web Ontology Language described in the Subsection 2.2.2. The choice of the language depends on the detail and complexity required to describe a specific domain.

2.2.1 Resource Description Framework Schema

RDF provides a way to express simple statements about resources, using properties and values. However, RDF itself simply creates a graph structure to represent data. RDF Schema [13, 2, 1] provides some guidelines about how to use this graph structure in a disciplined way.

The main idea of the RDF Schema is that it should clarify the semantic relationships between elements. All the schema information in RDFS is defined with RDF triples. The relationship between resources in RDF and schema resources are made with triples, as are relationships between resources. This feature makes particularly easy to provide a formal description of the semantics of RDF. Since everything in RDF is expressed as triples, even the schema information that provides context for the data on the Semantic Web can itself be distributed on the Semantic Web.

RDFS provides a system, somehow similar to an object-oriented programming language, that defines classes and properties, relates resources as instances of one or more classes, relates properties to more generic properties and organizes information in a hierarchical way.

A class in RDFS corresponds to the generic concept of a type or category. A class can be used to represent a collection of things with features in common. A resource that belongs to a class is called its instance. All the elements in an RDF document are members of the class rdfs:Resource the common ancestor class, even if it used implicitly. In RDF, a class of a resource is assigned with the rdf:type element. The value of this triple is a resource rdfs:Class. The definition of rdfs:Class is recursive:
rdfs:Class is the rdfs:Class of any rdfs:Class. There are a few more elements of
the RDF Schema classes. The rdfs:Literal is used to describe literal values, such
as strings and integers. It has a subclass, rdfs:Datatype which is the class of the
datatypes that has the data typing information. The rdfs:XMLLiteral is also a subclass
of rdfs:Literal and an instance of rdfs:Datatype and is the class of all XML literals.
The rdf:Property is a class used to define the attributes that describe a resource.

The relationship between classes is described by the element rdfs:subClassOf to
hierarchically relate the two classes. The meaning of this relationship is that any
instance of a subclass is also an instance of the super class. Hierarchies of classes
support inheritance of the property domain and range from a class to its subclass.
RDFS provides means to relate classes to one another by the subclass relationship.

In order to give meaning to the data it is necessary to describe the properties that links
the elements (the predicates of the triples). This schema language provides a simple
mechanism to relate properties similar to the one used with classes. The specialization
of two properties is described using the element rdfs:subPropertyOf. A property may
be subproperty of zero, one or more properties. All the features related to a property
are also applied to each of its subproperties.

Using RDF Schema it is possible to specify property constraints, providing important
information with two elements: rdfs:domain and rdfs:range. The rdfs:domain ele-
ment indicates that a particular property applies to a given class. In RDF, a property
description is, by default, independent and has global scope. With RDF Schema a
property can have no domain specified, being possible to extend the use of that property
definition to several situations. The rdfs:range element indicates the values of a
property as instances of a given class. It is not possible in RDFS to define a specific
property as having locally-different ranges, depending on the class of the resource it is
applied to. Any range defined for a property applies to all uses of that property.

Figure 2.5 shows the connection between an RDF and an RDF Schema. The blocks
are properties. The ellipses above the dashed line are classes and bellow the dashed
line are instances of the related class. The RDF resources relate with RDFS classes by
types. In the RDFS, classes and properties are hierarchically related and the properties
have constraints defined.
The RDF Schema language has several limitations to the represent ontological knowledge. The properties have a local scope. When the range of a property is defined, it stands for all classes. It is not possible to declare range restrictions to apply a property to some classes only. Special characteristics of properties, such as transitivity, uniqueness or inverse is not possible to define. It is not possible to create classes disjointed or combine classes using the Boolean operators, such as union, intersection and complement. Cardinality restrictions are also unavailable.

In summary, the RDF Schema provides a schema information as additional descriptions of resources but does not determine how the descriptions should be used by an application. It may also define constraints if the application interpreting those statements wants to treat them that way. All the RDF Schemata provide a way of state additional information, but if that information conflicts with the RDF data is up to the application to resolve it.

Figure 2.5: RDF and RDFS example
2.2.2 Web Ontology Language: OWL

The RDF Schema language helps to give meaning to the RDF data. However, the expressivity of this language is limited to hierarchical organization of classes and properties and to the simple constraints domain and range of a property.

OWL [1, 39, 4], the Web Ontology Language, is an RDF language, developed by the W3C that adds additional constraints to the ontology increasing the accuracy of implementations of a given vocabulary. It also allows additional information to be inferred about the data, though it may not be specifically recorded. OWL provides three sublanguages, with increasing expressiveness, each of which is designed to fulfil different needs of specific communities and users.

The OWL Lite is the simplest language and has a low formal complexity. It supports hierarchical classification and simple constraints. OWL Lite excludes enumerated classes, disjointness statements and cardinality above 1 (it only supports values of 0 or 1). These expressiveness limitations result in a minimal subset of language features that are easy to learn and implement. Implementations that only support OWL Lite vocabulary eases the interoperability between the OWL system with the RDFS models, databases or other reasoning tools.

The OWL DL (OWL Description Logic) supports the maximum expressiveness that preserves computational completeness and decidability. This means that all conclusions are computable in finite time. This language includes all the OWL language constructs but they can only be used under certain restrictions. The separation between elements (classes, properties and datatypes, etc) are required. Classes and properties must be explicitly typed as OWL classes and properties, respectively. Also, there are different types of properties in OWL: datatype properties and object properties. These two sets are disjointed, and some characteristics of the object properties (inverse of, inverse functional, symmetric and transitive) cannot be applied on datatype properties. These constraints are needed in order to support decidable reasoning.

The OWL Full is the entire language and is meant for users who want maximum expressiveness and syntactic freedom of RDF without computational guarantees. It combines the primitives of the RDF and RDF Schema with its own primitives. For
example, in OWL Full, a class can be treated simultaneously as a collections of individuals and as an individual in its own right. The advantage of OWL Full is that it is fully compatible with RDF. Any legal RDF document is also a legal OWL Full document and any valid conclusion in RDF/ RDF Schema is also a valid conclusion in OWL Full. The disadvantage of this language is that it becomes so powerful that it is undecidable.

When developing an ontology one needs to consider which sublanguage best suits the needs. OWL Lite is less expressive than OWL and the choice depends on the level of expressiveness required. OWL and RDF Schema can describe themselves using their own language. RDFS has some constructions that are very expressive, such as rdfs:Class, that can lead to decidability problems. Then, the choice between OWL DL and OWL Full mostly depends on the need of using the features of the RDF Schema. Using OWL Full will provide a less reasoning support.

The OWL documents are usually called OWL Onlotogies and are RDF documents. The root element is rdf:RDF and specifies a number of namespaces used in the document.

The document can be described using the element owl:Ontology that contains information about the ontology itself. It can provide information about the document version with the owl:versionInfo and it also has an imports section using the owl:imports, as shown in the example bellow. The imports section includes a rdf:Resource attribute that points to another OWL ontology and this element has a transitive property.

While namespaces are used for disambiguation purposes, imported ontologies provide definitions that can be used in the document.

```xml
<rdf:RDF
    xmlns:rdf="http://w3c.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3c.org/2002/07/owl#"
    xmlns:rdfs="http://w3c.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3c.org/2001/XMLSchema#">
<owl:Ontology rdf:about="http://example.org/ontology">
    <rdfs:comment>A simple ontology to a movie database.</rdfs:comment>
    <owl:versionInfo>1.2 08/2013</owl:versionInfo>
    <owl:imports rdf:resource="..."/>
</owl:Ontology>
</rdf:RDF>
```

Listing 2.5: An example of the import section.
Classes in OWL are defined using a `owl:Class`, a subclass of `rdfs:Class` and they express a concept. Every class in OWL is related to a set of individuals that are an extension of that class. The element `rdfs:subClassOf` states that a class is a subset of another. The element `owl:equivalentClass` defines the equivalence between classes. To state that instances of one class are not members of other classes, the element `owl:disjointWith` is used.

There are two predefined classes: `owl:Thing` and `owl:Nothing`. Every class is a subclass of `owl:Thing` and a superclass of `owl:Nothing`. These classes are used to assert facts about all or no instances.

The relationship between two different classes can be more complex than the subtyping allowed by `rdfs:subClassOf`. These relationships, based on set operations, are managed through a set of properties. The intersection of a class with one or more classes is defined with the element `owl:intersectionOf`. This operation is analogous to logical conjunction. In general it defines its subject class as being exactly equivalent to the intersection of all the classes that appear in the `owl:intersectionOf` element. The intersected classes do not have to be named classes, and be complex class descriptions themselves. The union of classes creates a class whose members combine properties of all classes being joined. An `owl:unionOf` statement describes an anonymous class which the class extension contains those individual that occur in at least one of the classes. This operation is analogous to logical disjunction. It is also possible to create a class that has all members of a specific domain that do not belong to a specific class using the element `owl:complementOf`. This operation is analogous to logical negation.

The members of a class can be explicitly enumerated using `owl:oneOf`. This construct says that the members are exactly those given. In order to create the set of members, the element `rdf:parseType = ‘Collection’` is used. A subelement is given for each member that is explicitly typed.

A class can also be described in terms of restriction on the property values that may occur for instances of the class. OWL has two kinds of property restrictions: value con-
constraints and cardinality constraints. These restrictions can be applied both to Datatype Properties and to Object Properties. Property restrictions have the following general form:

```xml
<owl:Restriction>
  <owl:onProperty rdf:resource="(some property)"/>
  <!-- precisely one value or cardinality constraint -->
</owl:Restriction>
```

Listing 2.6: An example of property restriction.

The element `owl:Restriction` contains the element `owl:onProperty` and one or more restriction declarations. The `owl:onProperty` element links a restriction to a particular property. The restriction statement should be a triple that represents the constraint on the property under consideration.

A value constraint puts constraints on the range of the property when applied to a particular class. The value constraint `owl:allValuesFrom` specifies an universal constraint and it is used to specify the class of possible values the related property can take. An existential constraint can also be specified by using the element `owl:someValuesFrom`. It defines a class that is the set of all objects such that at least one value for the specified class is an instance of the class. In order to specify a value for a property, as opposed to its class the element `owl:hasValue` is used. The value could be either an individual or a data value.

Cardinality constraints are used to restrict the cardinality of properties locally, in the class context. It makes a property have a specific number of values. The element `owl:minCardinality` specifies the minimum number of distinct values that instances can have for the property. Conversely, the `owl:maxCardinality` element specifies the maximum number of distinct values that instances of a class can have for the property. The element `owl:cardinality` specifies the exact number of distinct values that instances need to have for the property. This definition can also be achieved by combining the previous two cardinality constraints with the same value.
Properties

There are two distinct types of properties: Object Properties and Datatype Properties. The first relates objects to other objects while the second relates objects to datatype values. Since OWL does not have predefined datatypes, it allows the use of XML Schema data types. An object property is defined as an instance of `owl:ObjectProperty` and the datatype property as an instance of `owl:DatatypeProperty`. Both are subclasses of the RDF class `rdf:Property`. In OWL DL and OWL Lite, these two types are disjoint. But in OWL Full, since data values can be treated as individual, there are no differences between the two types.

From the RDF Schema, OWL inherits three elements. The `rdfs:subPropertyOf` hierarchically relates two properties. The `rdfs:domain` states that a property belongs to a specific class. And the `rdfs:range` assigns a specific class or data range to the object of a property.

To define the relationship between properties is used `owl:equivalentProperty` and `owl:inverseOf` elements. The `owl:equivalentProperty` states that two properties have the same property extension, that is the same domain and range. Notice that the equivalence property is different from the equality property. Equivalent properties have the same “values” (domain and range), but may have different meaning. Equality between properties should be expressed using the element `owl:sameAs`. Since properties have a direction, from domain to range, sometimes it is useful to define relations in both directions. The `owl:inverseOf` is used to define an inverse relation between properties. Using this element, the domain of one in the range of the other and the range of one is the domain of the other.

The `owl:FunctionalProperty` and `owl:InverseFunctionalProperty` are used to describe global cardinality constraints. The first element describes a property that can only have one unique value for each instance, that means that there cannot be two distinct values, \(y_1\) and \(y_2\), such the pairs \(x, y_1\) and \(x, y_2\) are both instances of the same property. The second element defines a property for which two different objects cannot have the same value.

The transitivity and symmetry can be defined using specific elements that are sub-
classes of owl:ObjectProperty. The owl:TransitiveProperty defines a transitive property, meaning that is a pair \((x,y)\) in an instance of \(P\) and the pair \((y,z)\) is also an instance of \(P\) then can be inferred that the pair \((x,z)\) is also an instance of \(P\). The owl:SymmetricProperty defines a symmetric property, meaning that if the pair \((x,y)\) is an instance of \(P\), then the pair \((y,x)\) is also a instance of \(P\). The domain and the range of a symmetric property are the same.

**Individuals**

OWL can also be used to relate individuals. Individuals are instances of one or more classes and are declared in as RDF. The assumption of “unique names” (different names refer to different things) is not possible on the Web because a thing can be referred in many ways. OWL needs an explicit statement to refer the relationship between individuals. It provides three elements for stating facts about the identity of the individuals.

The element owl:sameAs links an individual to another individual. It states that the two elements have the same identity. The element owl:differentFrom states that two instances are different individuals. To state the difference between individuals of a set the element used is owl:AllDifferent.

### 2.2.3 Comparing RDF Schema and OWL

The purpose of RDFS is to express the meaning of the relationships between classes and properties using a standard, RDF based format. Although OWL has a similar purpose and it is based on RDFS, it is more complex and expressive.

One of the most relevant differences between RDFS and OWL is the vocabulary. RDFS vocabulary is very restrict while OWL vocabulary is a larger vocabulary that can be used to state all the kinds of things.

The second major difference is the rigidity. In RDF Schema, one is free to state whatever one wants. A thing can be simultaneously a class and an instance. With OWL this is not legal. It imposes a more rigid structure to the data. OWL prescribes exactly how the vocabulary must used. RDFS provides more freedom (for instance, one can
model RDFS classes in RDFS) with the cost of creating potentially undecidable models.

The last key difference is the ability to import other ontologies that OWL has and RDFS does not. OWL also provides a variety of annotations that are useful to easily link data models together without losing coherence.

In conclusion, OWL provides a larger vocabulary making easy to say anything about a data model. It also allows to have a more rigid structure that eases the inference process. And, at last, OWL enables modularization; it is possible to have a set of ontology documents and only import the documents with the descriptions needed. However, these advantages take some effort to be familiarized with but, for small and simple ontologies, RDFS is used.

### 2.2.4 Common Vocabularies

As said before, the Linked Data initiative encourages the interlinking of data in order to increase its value and usefulness. In particular, it is recommended to use terms from well-known vocabularies whenever they are appropriated, rather than creating equivalent ones. The creation of new terms should only be done when the required terms lack in existing vocabularies. It should be noted that is also possible to mix several vocabularies.

**FOAF: Friend-of-a-Friend**

The Friend of a Friend (FOAF) [14, 20] is a community project that intends to provide a RDF vocabulary for expressing metadata about people, their interests, relationships, and activities. It was founded in 2000 by Dan Brickley and Libby Miller. The FOAF vocabulary is expressed using RDF and OWL and publishes its schema and specifications at the [http://xmlns.com/foaf/0.1](http://xmlns.com/foaf/0.1) namespace.

With FOAF it is possible to create the Semantic Web equivalent of a personal webpage, with the name, mailbox, homepage, and so on. This information is published as a linked data document that allows FOAF documents do link among then, creating a web of data that does not need a centralized database.

This vocabulary has a variety of terms to describe people, groups and documents.
Different types of applications can use or ignore different parts of FOAF. The main FOAF terms [15] are grouped in three categories:

- **Core**: the classes and properties of this category describe characteristics of people and social groups that are independent of time and technology. In the Table 2.2 this first column has the main elements of this category. The classes begin in capital letter and the properties are written in lowercase.

- **Social Web**: this category includes terms for describing Internet accounts, address books and other web-based activities. In the Table 2.2 this second column has the main elements of this category. The classes begin in capital letter and the properties are written in lowercase.

- **Linked Data utilities**: this category has terms that are used for educational and technical utility purposes.

![Figure 2.6: FOAF classes diagram.](image)

The Figure 2.6 shows how classes are related to one other. The blue rectangles are the “super classes”. The green ones are classes that are classified as stable. The yellow ones are classes that are still in testing. The following serialization is a simple example how FOAF is used to describe a person. The example is related to the example in the Figure 2.3.
<rdf:RDF
    xmlns:rdf="http://w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://w3c.org/2000/01/rdf-schema#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:ex="http://example.org/">
  <foaf:Person>
    <foaf:name>Quentin Tarantino</foaf:name>
    <foaf:firstName>Quentin</foaf:firstName>
    <foaf:surname>Tarantino</foaf:surname>
    <ex:bday>27/03/1963</ex:bday>
    <ex:directorOf>Pulp Fiction</ex:directorOf>
    <foaf:knows>
      <foaf:Person>
        <foaf:name>John Travolta</foaf:name>
        <foaf:firstName>John</foaf:firstName>
        <foaf:surname>Travolta</foaf:surname>
        <ex:bday>18/02/1954</ex:bday>
        <rdfs:seeAlso rdf:resource="http://example.org/Person/JT"/>
      </foaf:Person>
    </foaf:knows>
  </foaf:Person>
</rdf:RDF>

Listing 2.7: Describing a person with FOAF.
### Table 2.2: FOAF classes and Properties.

<table>
<thead>
<tr>
<th>FOAF Core</th>
<th>FOAF Social Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>nick</td>
</tr>
<tr>
<td>Person</td>
<td>mbox</td>
</tr>
<tr>
<td>name</td>
<td>homepage</td>
</tr>
<tr>
<td>title</td>
<td>weblog</td>
</tr>
<tr>
<td>img</td>
<td>jabberID</td>
</tr>
<tr>
<td>depiction</td>
<td>mbox_sha1sum</td>
</tr>
<tr>
<td>familyName</td>
<td>interest</td>
</tr>
<tr>
<td>givenName</td>
<td>topic_interest</td>
</tr>
<tr>
<td>knows</td>
<td>topic_interest</td>
</tr>
<tr>
<td>based_near</td>
<td>workplaceHomepage</td>
</tr>
<tr>
<td>Agent</td>
<td>workInfoHomepage</td>
</tr>
<tr>
<td>age</td>
<td>schoolHomepage</td>
</tr>
<tr>
<td>primaryTopic</td>
<td>publications</td>
</tr>
<tr>
<td>Project</td>
<td>currentProject</td>
</tr>
<tr>
<td>Organization</td>
<td>pastProject</td>
</tr>
<tr>
<td>Group</td>
<td>account</td>
</tr>
<tr>
<td>member</td>
<td>OnlineAccount</td>
</tr>
<tr>
<td>Document</td>
<td>accountName</td>
</tr>
<tr>
<td>Image</td>
<td>accountServiceHomepage</td>
</tr>
<tr>
<td></td>
<td>PersonalProfileDocument</td>
</tr>
<tr>
<td></td>
<td>tipjar</td>
</tr>
<tr>
<td></td>
<td>sha1</td>
</tr>
<tr>
<td></td>
<td>thumbnail</td>
</tr>
</tbody>
</table>

**DCMI - Dublin Core Metadata Initiative**

Dublin Core Metadata Initiative (DCMI) [28] is an initiative to create a vocabulary to describe web (images, webpages, videos, etc.) and physical resources (books, artwork, etc.). The DCMI has an abstract model built on top of RDF and specifies an abstract syntax for metadata records independent of particular encoding syntaxes.

In the **DCMI Resource Model** the description of a thing is made up of one or more *resource* about one, and only one, *resource*. Each *resource* is described using one or more *property-value* pairs. Each pair is made up of one *property* and one *value*. Each *value* is a *resource* that can be a *literal* value or *non-literal* value which is a physical, digital or conceptual entity. A *literal* is an entity, usually a string, together with
an optional language tag or datatype to denote the resource.

The Description Set Model allows the description sets, which means, describes single resources that are related in some way. It is made up of one or more statements about one and only one resource and zero or one described resource URI that is the URI that identifies the described resource. The resource are built in a similar way as in the previous model.

There are fifteen "basic" elements in the Dublin Core vocabulary, listed in the Table 2.3.

<table>
<thead>
<tr>
<th>Elements</th>
<th>Refinements</th>
<th>Encodings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>Abstract</td>
<td>Is refered by</td>
</tr>
<tr>
<td>Title</td>
<td>Access rights</td>
<td>Is replaced by</td>
</tr>
<tr>
<td>Creator</td>
<td>Alternative</td>
<td>Is required by</td>
</tr>
<tr>
<td>Contributor</td>
<td>Audience</td>
<td>Issued</td>
</tr>
<tr>
<td>Publisher</td>
<td>Available</td>
<td>Is version of</td>
</tr>
<tr>
<td>Subject</td>
<td>Bibliographic citation</td>
<td>License</td>
</tr>
<tr>
<td>Description</td>
<td>Conforms to</td>
<td>Mediator</td>
</tr>
<tr>
<td>Coverage</td>
<td>Created</td>
<td>Medium</td>
</tr>
<tr>
<td>Format</td>
<td>Date accepted</td>
<td>Modified</td>
</tr>
<tr>
<td>Type</td>
<td>Date copyrighted</td>
<td>Provenance</td>
</tr>
<tr>
<td>Date</td>
<td>Date submitted</td>
<td>References</td>
</tr>
<tr>
<td>Relation</td>
<td>Education level</td>
<td>Replaces</td>
</tr>
<tr>
<td>Source</td>
<td>Extent</td>
<td>Requires</td>
</tr>
<tr>
<td>Rights</td>
<td>Has format</td>
<td>Rights holder</td>
</tr>
<tr>
<td>Language</td>
<td>Has part</td>
<td>Spatial</td>
</tr>
<tr>
<td></td>
<td>Has version</td>
<td>Table of content</td>
</tr>
<tr>
<td></td>
<td>Is format of</td>
<td>Temporal</td>
</tr>
<tr>
<td></td>
<td>Is part of</td>
<td>Valid</td>
</tr>
</tbody>
</table>

Table 2.3: Dublin Core elements.

Nowadays, the Dublin Core metadata is categorized in four levels of interoperability. The levels built on each other and the third and fourth levels are still experimental.

1. Level 1: Shared terms definitions: the set of fifteen Dublin Core elements provide a vocabulary of concepts based on natural language definitions. The use of URIs
is not a requirement neither specified domains and ranges. Interoperability, in this level, is not a priority.

2. Level 2: Formal semantic interoperability: in this level, the interoperability between applications is based on the shared formal RDF model. The properties and the classes defined in the DCMI Metadata Terms have been defined for compatibility with Linked Data principles. Semantics, in this sense, does not refer no well-formed natural language definitions; it refers to formally stated relationships between terms and rules to produce logical inference. The use of URIs is mandatory so the conformance with formally specified domains, ranges and sub-property relations. This is the level that appears to be growing the fastest.

3. Level 3: Description Set syntactic interoperability: in this level, applications are compatible with the Linked Data model. In addition, share an abstract syntax for validations of records, the “description set”

4. Level 4: Description Set Profile interoperability: the records are exchanged between applications and in addition, a common set of constraints reflect the shared model.
Chapter 3

Web of Data

The amount of information available in the web is growing exponentially. People access to this information mainly through web documents. This documents, produced by humans to humans, are interlinked creating a Web of Documents. However, most of this information is inaccessible to machines. Websites contain a lot of information that is irrelevant to machines and that needs to be filtered.

The idea of Web of Data originated as result of the limitations of the Web of Documents and the countless structured data sets spread all over the world with all types of information. Typically, a data set contains knowledge about a particular domain, such as music, scientific research, books, and are property of an institution. Making those data sets interlinked, a machine can process the structured data, gathering knowledge about entities and domains. The result would be a massive and freely accessible knowledge net that could be the foundation of a new generation of services and applications.

The Web of Data is built of data weaved together as RDF triples that expresses the relationships among them. This vision is based on Linked Data: a set of techniques for the publication of structured data on the Web using standard formats and interfaces[10]. It is based in two fundamental standards: Uniform Resource Identifiers (URIs) and the Hypertext Transfer Protocol (HTTP). An important property of Linked data is that it may be easily combined with other Linked Data to create new knowledge.

With Linked Data, the World Wide Web becomes a global database: the Web of Data. Linked Data can be queried from multiple sources and dynamically combined. This
is almost impossible to do with traditional data management technologies, such as relational databases.

3.1 Data Integration

Traditionally, a tool understands only a specific type of messages and protocols and knows how to display that information to the user. The emergence of the Web changed how most people used the Internet. The Web is a standardized infrastructure that allows web application developers to work behind a facade that separates them from details of how application data is transmitted and focus on how their applications appear to users. Leveraging on this standardization, many new applications became available to the users. However, an application that combines data in new ways and allows users to make connections and understand hidden relationships is hard to achieve. The process of building that type of application is highly specialized, with each application using ad-hoc techniques for collecting and integrating information.

3.1.1 Traditional Data-Modeling Methods

There are many ways to model data and some of them are very well understood and mature.

Tabular Data

Tabular data is the simplest kind of data set. The data is kept in a table, as a spreadsheet or an HTML table. Its advantage is to be very simple to read and manipulate. Consider Table 3.1 listing the movies playing in a particular city.

Data in a table is usually easy to display, sort, print and edit. The placement of the data in columns and rows gives each piece of data a particular meaning. There are semantics in a table; the row and column in which the data is stored explains what the name means to the person using the data. However data stored in this way has limitations. Let's consider the Table 3.2.

Adding more information to a column sorting the information does not capture the
With this new type of data modeling it is possible to execute sophisticated queries. The
previous example and the Figure 3.2 the tables created.

A relational database allows multiple tables to be joined in a standardized way. To store
data in a relational database it is necessary to define a schema. The Figure 3.1 represents
the schema needed to store in a relational database the movie list from the
previous example and the Figure 3.2 the tables created.

With this new type of data modeling it is possible to execute sophisticated queries. The
deeper meaning of the text. In the example, the session hours were added to the
column Session. But now, a program does not understand that an individual field is
used to store multiple distinct information values.

### Relational Data

Relational databases are widely used in all kinds of applications in every industry. They
are very mature as result of years of research and optimization. Relational databases
are a very fast and powerful tools for storing large sets of data where the data model is
well understood.

A relational database allows multiple tables to be joined in a standardized way. To store
data in a relational database it is necessary to define a schema. The Figure 3.1 represents
the schema needed to store in a relational database the movie list from the
previous example and the Figure 3.2 the tables created.

<table>
<thead>
<tr>
<th>Title</th>
<th>Theater</th>
<th>Genre</th>
<th>Rate</th>
<th>Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Madagascar</td>
<td>UCI</td>
<td>Animation</td>
<td>6.8</td>
<td>Sat, Sun</td>
</tr>
<tr>
<td>The Dark Knight</td>
<td>UCI</td>
<td>Action</td>
<td>8.9</td>
<td>Thu, Fri</td>
</tr>
<tr>
<td>Snatch</td>
<td>Rivoli</td>
<td>Thriller</td>
<td>8.0</td>
<td>Thu, Fri</td>
</tr>
<tr>
<td>Finding Nemo</td>
<td>UCI</td>
<td>Animation</td>
<td>8.1</td>
<td>Sat, Sun</td>
</tr>
<tr>
<td>Wall-e</td>
<td>UCI</td>
<td>Animation</td>
<td>8.5</td>
<td>Sat, Sun</td>
</tr>
<tr>
<td>Pulp Fiction</td>
<td>Rivoli</td>
<td>Thriller</td>
<td>9.0</td>
<td>Thu, Fri</td>
</tr>
</tbody>
</table>

Table 3.1: A table with the movies playing this week.

<table>
<thead>
<tr>
<th>Title</th>
<th>Theater</th>
<th>Genre</th>
<th>Rate</th>
<th>Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Madagascar</td>
<td>UCI</td>
<td>Animation</td>
<td>6.8</td>
<td>Sat(16), Sun(16)</td>
</tr>
<tr>
<td>The Dark Knight</td>
<td>UCI</td>
<td>Action</td>
<td>8.9</td>
<td>Thu(21), Fri(19/22)</td>
</tr>
<tr>
<td>Snatch</td>
<td>Rivoli</td>
<td>Thriller</td>
<td>8.0</td>
<td>Thu(21), Fri(19/22)</td>
</tr>
<tr>
<td>Finding Nemo</td>
<td>UCI</td>
<td>Animation</td>
<td>8.1</td>
<td>Sat(16), Sun(16)</td>
</tr>
<tr>
<td>Wall-e</td>
<td>UCI</td>
<td>Animation</td>
<td>8.5</td>
<td>Sat(16), Sun(16)</td>
</tr>
<tr>
<td>Pulp Fiction</td>
<td>Rivoli</td>
<td>Thriller</td>
<td>9.0</td>
<td>Thu(21), Fri(19/22)</td>
</tr>
</tbody>
</table>

Table 3.2: A table with the movies playing this week with hours.
semantics of data also have been made more explicit. The meanings of the values are described by the schema; in the example table some types of entities modeled, such as “Movie” and “Session”, and they have a relationship between them. Each piece of data is labeled with what it means by virtue of the table and column that it is in.

But data is not static and the Web is characterized by changing types of data. To introduce more types of data in a relational database by adding more tables could introduce problems to it. Probably it is necessary to change the database schema. This process is called schema evolution[37]. Plus, in addition to having to migrate as the data evolves, relational databases schemata can get very complicated, specially if it has to deal with many different kinds of data.

### 3.1.2 Semantic Relationships

Another way to display data is to make knowledge explicit by displaying data in a parametrized way. The data becomes described alongside the property that defines it. This is the essence of semantic data modeling: flexible schemata where the relationships are described by the data itself.

Using someone else’s relational data could be a challenge because it is necessary to understand how the tables are related to each other. This kind of information, data about the data representation, is called metadata and represents knowledge about how the data can be used. It represents, in an explicit way, foreign key relationships and,
implicitly, the logic of the queries. However, it is usual that data is archived or published without metadata and recover that information can be very hard.

If data is represented in a flexible model, schema-independent, it is easy integrate new information without the concerns related to relational databases. Since the schema is explicit and is stored as data, this model allows schema interrogation in the same way as data. In the Section 2.1 is explained how to represent “self-describing” data in this manner and how to query semantic data.

3.2 Linked Data

Linked Data relies on a common data model that eases the process of combining data from different sources. Its self-descriptive nature enables cooperation without coordination. To illustrate this property consider a table as in Table 3.3. To understand the table it is needed to know the meaning of each column. However, sometimes the column header is not enough and the meaning is lost. A more descriptive column header could not be enough. The ideal is data describing the data – metadata – as shown in the Table 3.4 This information provides some context to the data to anyone.
that wants to use it.

<table>
<thead>
<tr>
<th>Day</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/2012</td>
<td>4</td>
</tr>
<tr>
<td>1/2/2012</td>
<td>8</td>
</tr>
<tr>
<td>1/3/2012</td>
<td>14</td>
</tr>
<tr>
<td>1/4/2012</td>
<td>15</td>
</tr>
<tr>
<td>1/5/2012</td>
<td>19</td>
</tr>
<tr>
<td>1/6/2012</td>
<td>23</td>
</tr>
</tbody>
</table>

Table 3.3: Example of table listing temperatures over a time period.

Table 3.4: Example of schema information related to Figure 3.3.

Linked data can provide the schema information that the data needs and also allow to publish the data itself in an open, extensible format. It also provides a way to link to other related data, anywhere on the Web, in an unambiguous way. Web addresses and dereferenceable identifiers are used both for schema and data itself. The Web addresses provide a way to get the data and related schema. The user only needs to follow the links.

3.2.1 Linked Data Principles

The Linked Data Principles[6] are a set of best practices for publishing and interlinking structured data on the Web. These practices were introduced by Tim Berners-Lee and are the following:

1. Use URIs as names for things.

2. Use HTTP URIs 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 first principle deals with identifying concrete or abstract things. The technology used to provide that identifier is the Uniform Resource Identifier. URIs used to name things in Linked Data are generalized version of URLs, used to locate web pages in the browser. A URI is a universally unique name; a URL is a special type of URI that resolves on the Web.

The second principle states that anyone must be able to look up the names of things. Using HTTP URIs it is possible to make the names resolvable on the Web. URLs combined with application protocols locate web pages unambiguously providing means to everyone retrieve the same document from the web when typing a specific URL.

The third principle says to allow the identifiers to resolve on the Web. When creating a URI to name a thing it should either refer an existing Web resource or make up one new. URIs must resolve to useful description about the named thing.

The fourth principle makes the Linked Data “linked”. The data becomes Linked Data when it links to related resources. Using HTTP URIs to publish data, other people can also link that data. The ability to follow these links allows people to surf in the Web of Data like surfing the traditional Web of Documents.

Linked Data is built directly on the Web architecture and, by combining simplicity with decentralization and openness, eases the task of sharing data on global scale.

### 3.2.2 Linking Open Data Project

The Linking Open Data Project[19] is a community activity started in 2007 by the World Wide Web Consortium’s Semantic Web Education and Outreach (SWEO) Interest Group. The project’s stated goal is to “make data freely available to everyone”.

The collection of published Linked Data is referred to as the “LOD cloud”[18, 3]. The Figure 3.3 is an attempt at visualizing the LOD cloud.

The last visualization of the LOD cloud is from September 2011. The cloud has grown
so large that no attempt was made since that date. The cloud has more than 300 datasets from various domains, such as Media, Publications and Life Sciences. All the data is available for use by developers. From the latest statistics, the cloud has more than 31 billion data items and about 500 million links between them.

### 3.3 Software Tools

There is a wide range of systems described in the literature that can be used to publish existing relational databases as Linked Data. In this section a closer look is taken to some of these applications and also to some frameworks used for developing Linked Data tools.

**Apache Jena**

Apache Jena[25] is an open-source Java implementation of the core of semantic web, developed by HP Laboratories, providing a large set of tools and Java libraries to ease the creation of semantic web and linked data applications. This framework includes:
1. an API for reading, processing and serializing RDF data in several formats.

2. an ontology API that handles RDF Schema and OWL ontologies.

3. a rule-based inference engine that enables reasoning between RDF and the OWL data sources.

4. two kinds of storage: persistent storage, such as SDB and TBD triplestores, that allows the efficiently storage of a large number of RDF triples or in-memory storage.

5. a query engine using SPARQL.

6. tools for publishing RDF data, with a variety of protocols, such as SPARQL.

Jena stores information as RDF in directed graphs. Using its API one can add, remove and manipulate data in these graphs, as well as store them in triplestores and publish them in SPARQL access points. Jena is the combination of several “sub-systems” as depicted in Figure 3.4.

The RDF triples and their components are accessed using the Jena’s RDF API. To represent a whole graph is used the abstraction Model and a triple which is a Statement. An RDF resource (named with URI or anonymous) is used the Resource and Literal is used to represent data values (strings, numbers, dates, and so on). This API eases adding or removing triples to graphs and finding triples that match particular patterns. RDF datasets can be also read from external sources (files or URLs) and be serialized in a specific form.

The RDF graph is stored in a simple abstraction that allows Jena to use a variety of different storage strategies. It can store a graph as an in-memory store, in a SQL database or as a persistent store using a custom based tuple index. The graph interface also allows that other stores to be connected to Jena.

Semantic rules of RDF, RDF Schema and OWL can be used to infer information that is not explicitly in the graph. Jena’s Inference API provides the means to make these entailed triples appear in the store. This specific API provides a number of rule engines using custom rules, created by users, or using built-in rules for OWL and RDFS or connecting to an external reasoner to perform the same job with other reasoning algorithms.
Jena also provides the Ontology’s API that handles both RDFS and OWL. Jena follows the collection of standards that define the semantic web technologies. This includes the query language with all of the publish standards, revisions, updates and under-development areas. The SPARQL’s API handles with both of query and update features of the interrogation language.

The features described above are typically accessed by applications, through the Java API. In order to publish data in the Web, Jena provides Fuseki, a data publisher server that presents and updates data from RDF models over web, using SPARQL and HTTP.
Sesame

Sesame [32] is an open-source Java framework for storing, querying and reasoning with RDF and RDFS. It can be used embedded in applications or in a standalone server mode, like a traditional database with multiple applications connecting to it. It provides tools to parse, interpret, query and store RDF data.

![Sesame Architecture](http://www.openrdf.org/)

**Figure 3.5:** Sesame Architecture.
Adapted from http://www.openrdf.org/

Similar to Apache Jena, Sesame has also a modular architecture that allows combining capabilities as needed. The Figure 3.5 shows the framework architecture [33].

The SAIL API (Storage And Inference Layer) is an internal Sesame API that abstracts from the storage format used and provides reasoning support. SAIL implementations can be combined providing functionalities such as caching and concurrent access handling.
On top of the SAIL API is the Sesame’s functional modules (SeRQL, RQL and RDQL query engines), the admin module and the RDF export module. These modules are accessed through the Sesame’s Access APIs that is divided in two separated parts: the Repository API and the Graph API. The Repository API provides high-level access to the Sesame repositories, handling all the details on client-server communication. Since the repositories can be local or remote, this API eases the access, the interrogation and updates of the repositories. The Graph API provides means to manipulate RDF in a more sensitive way, such as remove or adding individual statements or the creation of modes directly from the code. The Repository API and the Graph API complement each other in functionality and are usually used together.

The Access APIs provide direct access to the Sesame’s functional models to an applications or to the Sesame server. It provides the HTTP access to the Sesame’s API.

**D2RQ Platform**

D2RQ Platform[12] is implemented as a Jena graph, used to access relational databases as virtual, read-only RDF graphs. Using the virtual access to the relational content the replication of data is avoided. Using this platform is possible to query a non-RDF database using the query language SPARQL and access the content as Linked Data over the Web. It also creates custom dumps of the relational contents in RDF format in order to be loaded into a triplestore. There are three main components, as shown in the Figure 3.6 in this tool: the D2RQ Mapping Language, the D2RQ Engine and the D2R Server.

- D2RQ Mapping Language is a declarative language; it maps a relational database schemata to RDF vocabularies and OWL ontologies. This mapping defines the virtual RDF graph that contains the information from the database, in a similar way to the concept of views in SQL. The mapping can be written by hand or using a generate-mapping tool that creates a skeleton of the map using the database schema. The mapping uses database tables as classes and the columns as the properties of the classes.

- D2RQ Engine integrates with Apache Jena to process SPARQL over the relational databases.
• D2R Server is a port of viewing Linked Data over the web, providing “browsable” content in RDF format, human readable, which one can navigate through. It also provides the SPARQL endpoint.

The D2RQ Platform has an extension, the D2RD/Update[22], that extends the application and enables the execution of the SPARQL/Update statements (INSERT and DELETE) on the mapped data. This extension uses the constraints information from the relational database schema and operates according these constraints.

**OpenLink Virtuoso**

The OpenLink Virtuoso[40] is the open source edition of Virtuoso. This software is a cross platform SQL-ORDBMS (Structured Query Language - Object-relational database management system) and Web Application Server hybrid, known as Universal Server, that provides SQL, XML and RDF data management in a single server process. The triplestore access is available through SPARQL and other protocols. The Figure 3.7 shows the OpenLink Virtuoso architecture and how the different modules interact.
The SQL mapping to RDF[23, 24] is commonly referred as “Linked Data Views”. It enables the exposure of pre-existing accessible relational data as Virtual RDF graphs. The data is accessible through SPARQL queries or SPASQL (SPARQL within SQL). From this process results RDF datasets without physical regeneration of relational data.

The SPARQL to SQL translator recognizes triple patterns that refers to graphs and translates them to SQL. This method is used also by other applications such as D2RQ or Squirrel RDF. The key feature presented by Virtuoso is that it can process a query for which some triple patterns will go local or remote relational data and some to local physical RDF triples. Other feature brought by Virtuoso is the integration with SQL; since SPARQL and SQL share the same run time and query optimizer, the decisions are made with the best knowledge of data and its location. This is very important when mixing triples and relational data or when dealing with relational data distributed across several databases.

Generally speaking, any relational schema can be mapped into RDF. The primary and foreign keys are converted into IRIs, assigning a predicate IRI to each column and a
\texttt{rdf:type} predicate for each row linking that column to a class IRI that corresponds to the table. A triple with the primary key will be the subject, the column IRI the predicate and the column's value the object. This is a simple and often good approach, however it is very restrictive.

Virtuoso offers Data Model Storage Providers for Sesame and Apache Jena that provides means to ease the native persistence of RDF data via Virtuoso's Quad Store. These providers give better performance than those that persist RDF in SQL form because the overhead between the RDF graph and the relational model is eliminated.
Chapter 4

DaPress

4.1 DaPress Architecture

DaPress is a tool, developed with the Apache Jena Framework, that works as an intermediary between a Semantic Web client and a relational database. The tool extracts selected data from a relational database, transforms it into RDF and stores the generated triples in a specialized triplestore using a relational database\(^1\). Figure 4.1 depicts the DaPress architecture. The tool aggregates three components:

- **Manager** is the module responsible for loading the configurations files, opening the database connections and controlling the other modules.

- **Loader** implements the two mapping algorithms described in Section 4.2. This module is in charge of converting data into RDF, RDFS and OWL and store it in the triplestore.

- **Access Point** provides the SPARQL interrogation point. It is a specialized *servlet* providing a web service.

Mainly for testing purposes, there is also a simple web page for submitting SPARQL queries using a web browser, as shown in Figure 4.2. In the web page there are also some example queries and a UML schema of the data stored.

\(^1\)The source database and the triplestore may share the same database management system
Figure 4.1 also illustrates how the control (dashed arrows) and the data (full arrows) flow through the tool. Initially, the loading process is started by the Manager using the data in the configuration files. This operation is repeated periodically, in a frequency determined by the user. When the Loader receives the information passed by the Manager, it executes queries to the external relational database. With the result of those queries, the Loader creates the RDF and OWL models and store them in the triplestore. Later on, when a client makes a query through DaPress, the request is handled by the Access Point that interrogates the model stored in the triplestore.

The corner stone of DaPress is the mapping algorithms that converts relational data into RDF, RDF Schema and OWL driven by an XML configuration file. The following two subsections detail both the algorithms and the document type of the configuration files.
4.2 Mapping Algorithms

This section presents the mapping algorithms of DaPress. For sake of clarity the algorithm for creating plain RDF triples from the content of the relational database is separated from the algorithm for creation of the ontologies from the schema of the relational database. However, both algorithms work together and they consume data provided by the XML configuration file and data retrieved from the relational databases, using the SQL queries. Both algorithms produce a model, i.e. a collection of RDF triples. In DaPress these two modes are merged in a single one and stored in the same triplestore.
4.2.1 RDF Mapping Algorithm

The RDF mapping algorithm receives as input configuration data and relational data and produces as output a model – a set of RDF triples created with Jena.

In the Algorithm 1 the input is provided by a collection of functions. The inputs with the prefix selected are the result from the configuration data, returning information on the table.

- selectedTableNames returns a list with the table names that are intended to map.
- selectedFieldNames returns a list with the field names, for each table, that are intended to map.
- selectedResourceTypeName returns the type of the resource previously defined.
- selectedPropertyTypeName returns the value of the current field name.
- selectedRangeTypeName returns the value of the range defined for the current field name.

In contrast, the inputs with the prefix get correspond to data coming from the relational database which is the result of a query.

- getId maps the table names to list of ids.
- getValue maps the field and id pair to values.

Functions with the prefix make correspond to methods provided by the Jena API to create RDF elements.

- makeResource creates a new resource given an URI namespace, a type and an ID.
- makeProperty creates a new property using a namespace and a unique name to concatenate.
- makeLiteral creates a literal value, with the value given by string.

The algorithm 1 iterates over the selected tables and, for each one, retrieves their identifiers from the database. For each identified record, using the makeResource it creates a resource with the identifier retrieved. This function also creates the specific
Algorithm 1: RDF Mapping algorithm

Input: selectedTableNames(), selectedFieldNames()
Input: selectedResourceTypeName(), selectedPropertyName()
Input: selectedRangeTypeName()
Input: getIds(), getValue()
Output: model

model ← ∅

for tableName ∈ selectedTableNames() do
    for id ∈ getIds(tableName) do
        type ← selectedResourceTypeName(tableName)
        subject ← makeResource(type, id)
        for fieldName ∈ selectedFieldNames(tableName) do
            predicate ← makeProperty(selectedPropertyName(fieldName))
            value ← getValue(fieldName, id)
            range ← selectedRangeTypeName(fieldName)
            if range = NULL then
                object ← makeLiteral(value)
            else
                object ← makeResource(range, value)
            model ← model ∪ makeStatement(subject, predicate, object)
property from the RDF vocabulary, the \texttt{rdf:type}. With the \texttt{selectedResourceTypeName} value, the Ontology algorithm creates a new class (or retrieves the ontology class if already exists) and relates it to the resource created.

For each field of the current record, using the function \texttt{makeProperty}, a new property is created. The type of the property is defined by the value retrieved by the \texttt{selectedPropertyTypeName} function. According to the range defined to the property, that is given by the \texttt{selectedRangeTypeName}, a literal or a resource is assigned to the property. If the range is not defined, a literal is created with the value retrieved using the \texttt{getValue} function. Case the range is defined, a new resource is created using both the range and the value given by the \texttt{value}. The value field, in this case, can give origin to a class hierarchy, that is explained in the next sub-section. Finally, using the \texttt{subject}, the \texttt{property} and \texttt{object} created, a new statement is constructed using the \texttt{makeStatement} function and added to the model.

### 4.2.2 Ontology Mapping Algorithm

The algorithm for creating the ontology, presented in Algorithm 2, is similar to algorithm presented in the previous sub-section. It also creates RDF triples, but using properties defined in the RDFS and OWL vocabularies. To highlight the fact the model produced by this algorithm contains an ontology, it is labeled as \texttt{ontModel}.

The functions with the prefix \texttt{make} corresponds to methods provided by the Jena API to create ontologies.

- \texttt{makeOntClass} creates a new ontology class, that is represented by an URI, using a namespace together with an unique identifier.
- \texttt{makeDatatypeProperty} creates a new datatype property, represented by an URI, using a namespace with an unique identifier.
- \texttt{makeObjectProperty} creates a new object property also represented by an URI, using a namespace with an unique identifier.
- \texttt{makeDataRange} assigns a range to the datatype property, with the datatype value, usually XSD datatypes.
• `makeObjectRange` assigns a range to an object property, creating a new ontology class to the range value.

• `makeDomain` assigns a domain to a property using an ontology class.

• `makeSubClass` relates hierarchically two ontology classes.

• `makeDisjoints` given a list of classes, assures that one member of one class cannot be member of another class in the list.

Algorithm 2: Ontology Mapping algorithm

Input:
- `selectedTableNames()`, `selectedFieldNames()`.
- `selectedResourceTypeName()`, `selectedPropertyTypeName()`.
- `selectedValueAsType()`, `selectedRangeTypeName()`.
- `getIds()`, `getValue()`.

Output: `ontModel`

\[
\text{ontModel} \leftarrow \emptyset \\
\text{subClassList} \leftarrow \emptyset
\]

for `tableName \in \text{selectedTableNames()}` do

\[
\text{domain} \leftarrow \text{makeOntClass}(\text{selectedResourceTypeName}(\text{tableName})) \\
\text{ontModel} \leftarrow \text{ontModel} \cup \text{domain}
\]

for `fieldName \in \text{selectedFieldNames(tableName)}` do

\[
\text{range} \leftarrow \text{selectedRangeTypeName(fieldName)} \\
\text{if range} = \text{NULL then} \\
\quad \text{property} \leftarrow \text{makeDatatypeProperty(fieldName)} \\
\quad \text{makeDataRange(property, datatype)}
\]

\[
\text{else} \\
\quad \text{property} \leftarrow \text{makeObjectProperty(fieldName)} \\
\quad \text{makeObjectRange(property, makeOntClass(range))}
\]

\[
\text{makeDomain(property, domain)} \\
\text{ontModel} \leftarrow \text{ontModel} \cup \text{property}
\]

if `selectedValueAsType(fieldName)` then

\[
\text{for id} \in \text{getIds(tableName)} \text{ do} \\
\quad \text{value} \leftarrow \text{getValue(fieldName, id)} \\
\quad \text{subClass} \leftarrow \text{makeOntClass(value)} \\
\quad \text{makeSubClass(subClass, domain)} \\
\quad \text{ontModel} \leftarrow \text{ontModel} \cup \text{subClass} \\
\quad \text{subClassList} \leftarrow \text{subClassList} \cup \text{subClass}
\]

\[
\text{makeDisjoints(subClassList)}
\]

The Ontology algorithm has the same inputs of the RDF algorithm. Although most of the
data used to create classes and properties comes from the configuration file, the values from the database still have to be explored in cases where subclasses are encoded as auxiliary tables.

In Algorithm 2, for each selected tables, a new ontology class is created with the type name assigned to that table. This new class is added to the model and will be used as domain of the properties related to this type.

The selected fields of the current table are iterated and a property is created for each one, based on the range value. If the range is not defined, a new datatype property is created. The range of this property is also created, assigning a datatype value to the property. In contrast, if the range is defined, a new object property is created. To assign the range, a new ontology class is created using the range value. After the property is created, the domain is defined and the property is added to the model.

There is a special case when a field was selected as holding subclasses. In this case, the records of this table must be iterated and a new ontology class is created. These new classes are related to the domain class hierarchically and added to the model. The classes are stored in a list to be used in the creation of the disjointed classes.

4.3 Configuration Files

The DaPress uses a configuration file, provided by an XML Document that contains all the information required by the tool. It has four types of information:

- parameters to establish the relational database connections, such as type of database, database name, database location and credentials.
- general configuration with the SDB description file path and the delay for updates.
- selected resources which are the tables that the tool will map.
- selected properties which are the columns that the tool will map.

This type of document is formalized by an XML Schema definition, whose structure is depicted in the diagram of the Figure 4.3.
The most relevant part of this configuration file is about the resources. The element \textbf{Resources} contains a sequence of elements for each resource. Each resource has a group of attributes that defines the name of the resource, the namespace, the type and the related table in the relational database. It also provides information about the field that will be used as unique identifier. The name of the table is used for the SQL queries.
Figure 4.3: XML schema of DaPress configuration files
Each resource contains a set of properties. These properties have also a group of attributes defining the name of the property, the namespace, the related column in the database, the range if applied and a mandatory attribute. The mandatory attribute is a Boolean that allows the tool to know if the attribute needs to exist; if True, a `where` clause is added to the SQL query, stating that the current property (column in the query) must be Not Null.

A secondary configuration file is the SDB Description File. This file configures the connection to the triplestore and its path is defined in the DaPress in the main configuration file.

### 4.4 Algorithms and Configuration Examples

The following example illustrates how both algorithms manipulate the data available in the relational database and the resulting RDF graph. Both tables, Person and Location, have a one-to-many relationship. The Table 4.1 is a sample of the database.

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>gender</th>
<th>hometown</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maria</td>
<td>female</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Helen</td>
<td>female</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Ian</td>
<td>male</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Cristian</td>
<td>male</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Paul</td>
<td>male</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>Ana</td>
<td>female</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chicago</td>
<td>USA</td>
</tr>
<tr>
<td>2</td>
<td>Glasgow</td>
<td>UK</td>
</tr>
<tr>
<td>3</td>
<td>Madrid</td>
<td>Spain</td>
</tr>
<tr>
<td>4</td>
<td>London</td>
<td>UK</td>
</tr>
<tr>
<td>5</td>
<td>Porto</td>
<td>Portugal</td>
</tr>
<tr>
<td>6</td>
<td>Lisbon</td>
<td>Portugal</td>
</tr>
</tbody>
</table>

Table 4.1: Original database.

In order to execute the mapping a configuration file is needed. It can be consulted in Appendix B.

For each row in the Person table a node is generated. This node is identified by a unique identifier, the field name, and is the subject of the triples. Each column header will generate a new property. The name of that property can be either the name of the field or be renamed for a more accurate title. In this example, the column gender generates the property from the RDF vocabulary `rdf:type`, connecting a name to its gender. The column hometown relates a person’s name to a location. Since the location is related to other table, the value of that property, renamed to `x:isFrom`, is a resource
related to the table Locations. The next step is to build the ontology. With the table name a OWL class is create, named Person. The different values in the column gender can be related, to this class. For each different element, female and male a new OWL class is created and connect them, hierarchically, to the Person superclass. Since a person, using the table values, only can be female or male, is assumed that it cannot be the both genders at same time. So the property owl:disjointWith is created assuring that one person can only have a gender.

With the table Locations, the process is similar. The unique identifier in this table is the field id. A new resource is created for each row, if it was not created while processing the table Person. The other columns are used to create two new properties, x:name and x:countryOf. The values of these properties are literals (strings), retrieved from the table.

The prefix x is used to replace the full namespace URI http://example.org. After the mapping the RDF graph generated is illustrated in the Figure 4.4
Figure 4.4: Example of the algorithms application.
4.5 Validation

The validation of the DaPress is based on the experience gained while publishing existing relational database. For this purpose, three different databases with different sizes, structures and domains were selected

- **Authenticus** database[16] is part of a project\(^2\) with the same name that aims to develop a system to automatically assign publications and their authors to known Portuguese researchers and institutions. This system has several algorithms to perform the author name disambiguation and identification. One of the main outcomes of this project is a normalized and validated database of Portuguese publications, that is an apt example of the kind of data that should become available as Linked Data.

- Adapted Sakila database is a sample database developed by MySQL team. It intends to provide a standard schema that can be used for books, movies, articles and so forth. The database used an adaptation of this database. It is a video rental database, with films and actors, their participation, stores and customers.

- Employees database is a sample database, developed by MySQL team, that provides a large set of data.

Besides the mapping of these databases, another database was also converted to RDF. DBLP\(^3\) (Digital Bibliography & Library Project) is a computer science bibliography website, hosted in Germany by Universität Trier. It tracks the most important journals on computer science and proceedings papers of many conferences. This database is available in a XML dump format, updated almost daily. To generated a RDF/XML file from the XML dump, DBLP team also provides a XML Style Sheet (XSLT) and a Document Type Definition (DTD). The load of the generated file to the triplestore was done using the Jena API. The XML dump used in this process was from 18 June 2013.

Table 4.2 summarizes information on the databases used in the validation of daPress. The databases Authenticus, Sakila and Employees were transformed in RDF using the mapping algorithms in the Section 4. Some tables of the original databases were

\(^2\)https://authenticus.up.pt/
\(^3\)http://www.informatik.uni-trier.de/~ley/db/
excluded from the mapping, such as those related to user management or containing precomputed values to speedup frequently requested listings. In order to have a point of comparison between all the databases used they are measured as mySQL dumps before and after the mapping. Comparing the size of the databases before and after the mapping there is significant increase of the size of the dumps. In average, map a relational database increases in 378% the size of the database. This difference can be explained with the “explosion” of the number of records stored in the database. The number of triples (records) generated is related to the amount of tables and the amount of columns from each table that is mapped.

<table>
<thead>
<tr>
<th>Original Size</th>
<th>Authenticus</th>
<th>Sakila (adapted)</th>
<th>Employees</th>
<th>DBLP *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used Size</td>
<td>36.2 Mb</td>
<td>840 Kb</td>
<td>66 Mb</td>
<td>745 Mb</td>
</tr>
<tr>
<td>Number of Tables (original)</td>
<td>67</td>
<td>10</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Number of Tables (used)</td>
<td>13</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Number of SQL Records</td>
<td>496.857</td>
<td>7.278</td>
<td>1.076.634</td>
<td>3.596.355</td>
</tr>
<tr>
<td>RDF Size</td>
<td>153 Mb</td>
<td>2.13 Mb</td>
<td>300 Mb</td>
<td>4 Gb</td>
</tr>
<tr>
<td>Number of Triples Stored</td>
<td>2.062.838</td>
<td>38.033</td>
<td>4.802.273</td>
<td>49.011.398</td>
</tr>
<tr>
<td>% of Increasing Size</td>
<td>42%</td>
<td>259%</td>
<td>454%</td>
<td>550%</td>
</tr>
</tbody>
</table>

Table 4.2: Database specifications and size related results.

<table>
<thead>
<tr>
<th>Execution Time</th>
<th>Authenticus</th>
<th>Sakila (adapted)</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>194 min</td>
<td>3 min</td>
<td>491 min</td>
</tr>
<tr>
<td>Triples per second</td>
<td>177</td>
<td>211</td>
<td>163</td>
</tr>
<tr>
<td>Data per second</td>
<td>13 Kbps</td>
<td>12 Kbps</td>
<td>10 Kbps</td>
</tr>
</tbody>
</table>

Table 4.3: Time related results.

The Table 4.3 summarizes the time consumed by mapping algorithms. In average, the algorithms produced 11.7 Kb of data per second with an average of 184 triples per second.

Since one of our goals is to link the Authenticus data to the DBLP data, both RDF data are stored in the same triplestore, but in different graphs. The UML diagrams of the mapping of Authenticus and DBLP are available in Appendix C.
The machine where the mapping was processed is a Pentium 4 running at 2,4Gz with 8Gb of RAM. It is operated by Linux Mandriva 2009 with a 2.16.19 kernel. It should be noted that the machine available for these tests is rather old, with a single processor, thus the mapping should be faster on a multi-core machine. Nevertheless, the order of magnitude of these mapping times requires an incremental algorithm that is already planned.
Chapter 5

Conclusions and Future Work

This chapter presents the contributions of this MsC thesis to the publication of relational data as linked data, and identifies some of its shortcomings that may address in the future.

Part of the work described in this dissertation was presented on a paper at the Symposium Languages, Applications and Technologies SLATE'13 [17]. The best papers in this conference were selected for publication in the Computer Science and Information Systems journal (ComSIS). An extended version of the SLATE’13 paper, with content from this thesis, is being prepared to submit to ComSIS.

5.1 Conclusions

The main objective of this work was the creation of a tool for publishing the content of a relational database as Linked Data. This work involved four steps: (1) the definition of an algorithm to map relational data into RDF, (2) the definition of an algorithm to map relational schemata into ontologies (3) the creation an XML document type for configuration of the mapping process, (4) the development of a tool implementing the previous mapping algorithms, guided by the configuration document, and providing a SPARQL endpoint to interrogate the mapped data.

The approach used to map relational databases into Linked Data driven by an XML configuration document, is the major contribution of this work. The algorithms are
able to map relational tables with any kind of relationships (one-to-one, one-to-many, many-to-many) and either with a single field key or a set fields key. The use of XML documents proved to be a simple, flexible and expedite way to define and store mapping information. These ideas were incorporated in the DaPress system, and the design and implementation of this tool are also a relevant contribution of this work.

To validate the proposed approach the DaPress tool was used for publishing several preexisting relational databases. The most important test was with the publication of Authenticus database. This database served for testing the mapping algorithms. The validation results shown that there was a significative increase in the data size and that the time required for the relational data conversion was very high.

Linked Data published by this application is ready to be interconnected with similar or related sources, by sharing URIs of classes, properties and resources, or by relating them at the ontological level. This is the case of the RDF data of Authenticus that is interlinked with the DBLP that uses a common vocabulary for shared information. Publishing DBLP data with DaPress required loading a very large RDF file to the triplestore. The purpose of this exercise was to enable the future study of the interlinking of DBLP data with the data of Authenticus.

The conversion from relational data to RDF significantly increases the size of the data. Also, the time required to convert a medium size database, such as that of Authenticus, is also very high.

5.2 Future Work

The experience gained during the validation led to the identification of a number of issues.

The time necessary for the data conversion is too high to be used in regular updates. Making the algorithms incremental is fundamental to enable frequent updates. The updating process must not replicate the data already mapped in the database.

Even though using an XML configuration file is a simple, it requires some knowledge of XML. An administrative tool showing the tables and fields available on the relational database, enabling their selection and renaming for the mapping process, would sim-
plify configuration process.

Loading remote data in RDF format to a relational database is also a cumbersome process, as could be concluded from the attempt to use data from the DBLP database. These could be achieved by inverting the data conversion process, using the mapping configuration file.

Currently, RDF data of *Authenticus* and DBLP are already linked and in the same triplestore. Part of the future work will be to explore the possibilities open by this collection of linked data about the same domain. An immediate problem will be to identify common resources, such as researchers and articles, having in mind that different researchers may have the same name and different articles may have the same title.
Appendix A

Acronyms

API  Application Programming Interface
CSV  Comma-Separated Values
DBMS Database Management Systems
DCMI Dublin Core Metadata Initiative
DL   Description Logics
FOAF Friend of a Friend
HTTP Hypertext Transfer Protocol
IRI  Internationalized Resource Identifiers
LOD  Linking Open Data
NCNAME Non-colonized name
NoSQL Not Only Structured Query Language
OWL  Web Ontology Language
RDBMS Relational Database Management System
RDF  Resource Description Framework
RDFS Resource Description Framework Schema
RDQL RDF Data Query Language
APPENDIX A. ACRONYMS

**RQL** RDF Query Language

**SAIL** Storage And Interface Layer

**SPARQL** SPARQL Protocol and RDF Query Language

**SQL** Structured Query Language

**SQL-ORDBMS** Structured Query Language – Object-relational database management system

**SWEO** Semantic Web Education and Outreach

**SeRQL** Sesame RDF Query Language

**UML** Unified Modeling Language

**URI** Uniform Resource Identifier

**URL** Uniform Resource Locator

**URN** Uniform Resource Name

**W3C** World Wide Web Consortium

**XML** Extensible Markup Language

**XSD** XML Schema Definition

**XSLT** EXtensible Stylesheet Language Transformations
Appendix B

XML Configuration File
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY ex "http://exemplo.org/">

<tns:configs
  xsi:schemaLocation="http://www.example.org/TestConfigs.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:tns="http://www.example.org/Test">

  <tns:prefixes>
    <tns:prefixe name="ex">
      <tns:uri>http://example.org/</tns:uri>
    </tns:prefixe>
  </tns:prefixes>

  <tns:resources>
    <tns:resource name="persons" table="Person" namespace="&ex;"
        super="Person" identifier="name" type="persons">
      <tns:typeColumn>gender</tns:typeColumn>
      <tns:property name="isFrom" column="hometown" namespace="&ex;"
        range="locations" mandatory="false"/>
      <tns:property name="gender" column="gender" namespace="&ex;"
        mandatory="true"/>
    </tns:resource>

    <tns:resource name="locations" table="Locations" namespace="&ex;"
        identifier="id" type="location">
      <tns:typeName>locations</tns:typeName>
      <tns:property name="name" column="name" namespace="&ex;"
        <tns:property name="countryOf" column="country" namespace="&ex;"
          mandatory="false"/>
    </tns:resource>
  </tns:resources>

  <tns:databaseConfigs>
    <tns:username>myUsername</tns:username>
    <tns:password>myPassword</tns:password>
    <tns:url>jdbc:mysql://localhost/</tns:url>
    <tns:dbname>mySimpleDatabase</tns:dbname>
    <tns:driver>com.mysql.jdbc.Driver</tns:driver>
  </tns:databaseConfigs>

  <tns:generalConfs>
    <tns:delay>100000</tns:delay>
    <tns:sdbFile>/home/myhome/sdb.ttl</tns:sdbFile>
  </tns:generalConfs>
</tns:configs>

Listing B.1: XML Configuration Document Example
Appendix C

UML Diagrams

Figure C.1: Authenticus UML Diagram.
Figure C.2: DBLP UML Diagram.
Bibliography


