Publishing Linked data with DaPress

Chapter · August 2013

CITATION
1

READS
18

2 authors:

Teresa Costa
University of Porto
6 PUBLICATIONS **6** CITATIONS
SEE PROFILE

José Paulo Leal
University of Porto
97 PUBLICATIONS **288** CITATIONS
SEE PROFILE

Publishing Linked Data with DaPress*

Teresa Costa[1] **and José Paulo Leal**[2]

1 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto**
Porto, Portugal
up200101764@alunos.dcc.fc.up.pt

2 **CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto**
Porto, Portugal
zp@dcc.fc.up.pt

Abstract

The central idea of the Web of Data is to interlink the information available in the Web, most of which is actually stored in databases rather than in static HTML pages. Tools to convert relational data into semantic web formats and publish then as linked data are essential to fulfill the vision of a web of data available for automatic processing, as web content is currently available to humans. This paper presents DaPress, a simple tool to publish linked data on the Web, that maps a relational database to an RDF triplestore and creates a SPARQL access point. The paper reports the use of DaPress to publish the database of Authenticus, a system that automatically assigns publication authors to known Portuguese researchers and institutions.

1998 ACM Subject Classification H. Information Systems; H.2 Database Management; H.2.5 Heterogeneous Databases;

Keywords and phrases RDF, RDF Schema, Relational data; Semantic web

Digital Object Identifier 10.4230/OASIcs.SLATE.2013.67

1 Introduction

The World Wide Web has deeply changed the way information is produced, published and consumed. Nowadays it is trivial to produce a document in HTML format, publish it on a HTTP server and virtually anyone, anywhere on the planet, can access it using a web browser and benefit from its content. Anyone but not anything.

Information on the web is produced and formatted for humans. It is simple for a person to understand web content and navigate trough hyperlinks with a meaningful purpose. However, building 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, is still a challenge after more than a decade of research.

The goal of the semantic web is to open the vast amount of data available on the web to software processing. The first attempt was to markup with semantic annotations the content already available on web pages. The use of XML languages and the 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 is harder to provide semantic annotations to web apps and web services than it was to last century hand-made web pages.

* This work is in part funded by the ERDF/COMPETE Programme and by FCT within the FCOMP-01-0124-FEDER-022701 project.

2nd Symposium on Languages, Applications and Technologies (SLATE'13).
Editors: José Paulo Leal, Ricardo Rocha, Alberto Simões; pp. 67–81

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Fortunately, most web content is actually generated from databases. Thus, rather than to extract 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 the 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 of 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 trough data available from different sources. If URLs 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. Any 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 standards are not enough to build a web of linked data and must be complemented with semantic web technologies for a number of reasons.

Firstly, 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 hard to implement. In contrast, the *Resource Description Framework* (RDF) data has a simple and uniform structure – a collection of triples – and the schemata from the various databases are recorded also as RDF data using a special vocabulary – RDF Schema.

Secondly, the semantics of the data stored in relational database 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 "docent" from two different academic databases contains similar data, but that kind of reasoning is extremely difficult to automatise. The semantic web provides ontologies to describe a domain of data shared by different databases.

Lastly, 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. An approach to achieve it is to map relational databases into RDF data and web ontologies, while providing absolute control of this process to the data owner.

The motivation for the ongoing research presented in this paper is the development of a simple and flexible approach to publish the content of relational databases as linked data on the Web. The corner stone of the proposed approach is a system called DaPress that maps relational databases to RDF and RDF Schema based on a XML configuration file. Relational data from the source database is periodically loaded into the DaPress triplestores, which is accessible trough a SPARQL access point.

The remainder of this paper is organized as follows. Section 2 summarizes the concepts, languages and tools related to linked data. Section 3 presents the design and implementation details of DaPress, the proposed linked data publishing system. This approach was validated on the database of Authenticus, a system that automatically assigns publication authors to known researchers and institutions and the results are reported on Section 4. Section 5 concludes with a summary of the work presented in this paper and points to future developments of DaPress.

2 Linked Data

Linked Data is a methodology for publishing structured data based on two fundamental Web technologies: Uniform Resource Identifiers (URIs) and the HyperText Transfer Protocol (HTTP). The term *Linked Data* highlights the fact that this methodology establishes links among data from different sources, creating a *web of data*. Unsurprisingly, the concept of linked data is due to Tim Berners-Lee, the father of the World Wide Web, who introduced a set of basic rules for publishing data on the Web [4], namely:

- *Use URIs as names for things;*
- Use HTTP URIs so that people can look up those names;
- *When someone looks up a URI, provide useful information, using standards (RDF, SPARQL);*
- Include links of other URIs so that they can discover more things.

The resources, or "things" as Tim Berners-Lee calls them, are identified by URIs and these entities can be looked up simply by dereferencing the URI over the HTTP protocol. The HTTP protocol provides a simple and universal mechanism for retrieving resources or retrieving descriptions of entities.

By using HTTP URIs (or URLs) to identify entities, the HTTP protocol as retrieval mechanism and RDF data model to represent data, Linked Data builds on the general architecture of the Web.

The remainder of this section details the fundamental technologies used by Linked Data, such as RDF and RDF Schema, as well as related systems to publish relational data as RDF.

2.1 Resource Description Framework

The Resource Description Framework (RDF) [2, 10] is a framework for representing any kind of information available in the web. The RDF data model provides an abstract, conceptual framework for defining and using metadata, that has a graph-based data model, and is easy to process and manipulate by applications. It provides interoperability between applications that exchange machine-understandable information on the Web. Data in RDF format can be persistently stored in specialized repositories called triplestores, and retrieved using specialized RDF query languages, such as SPARQL. The interoperability of RDF data is supported by several serialization formats, both text and XML based.

2.1.1 Data Model

The basic element in RDF is a *statement*, a simple sentence with three parts – subject, predicate and object – expressing a relationship between things. The *subject* is a resource, the thing to describe, identified by an URI. The properties are a special kind of resource that describe relations between resources. A *property* specifies an aspect, characteristic, attribute or relation used to describe the resource. They are also identified by URIs. A specific resource together with a named property needs an *object*, in order to construct a statement. The object can be either a resource or an atomic value, named *literal*. Being composed of tree parts, RDF statements are also known as *triples*. A collection of triples forms a graph where the set nodes is given by subjects and objects of triples, and the arcs that connect them are given by predicates.

Table 1 Simple example of table-based tripes representation.

Subject	Predicate	Object
.../Person/QuentinTarantino	.../name	"Quentin Tarantino"
.../Movie/PulpFiction	.../name	"Pulp Fiction"
.../Person/QuentinTarantino	.../director	.../Movie/PulpFiction

Lets consider a simple example to show two different ways of represent a statement.

Quentin Tarantino is the director of Pulp Fiction.

Table 1 shows the triples extracted from previous phrase where ellipsis replace a common URL prefix for sake of terseness. Note that the concepts "Quentin Tarantino" and "Pulp Fiction" where replaced by URIs, as was the "is the director" property. By the cultural context, it is known that Quentin Tarantino is a person's name and Pulp Fiction a movie title. Using that information the other two statements assign a textual representation to both the subject and object of the previous sentence.

Figure 1 is a graph-based and equivalent representation of the same three statements. 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 a semantic net.

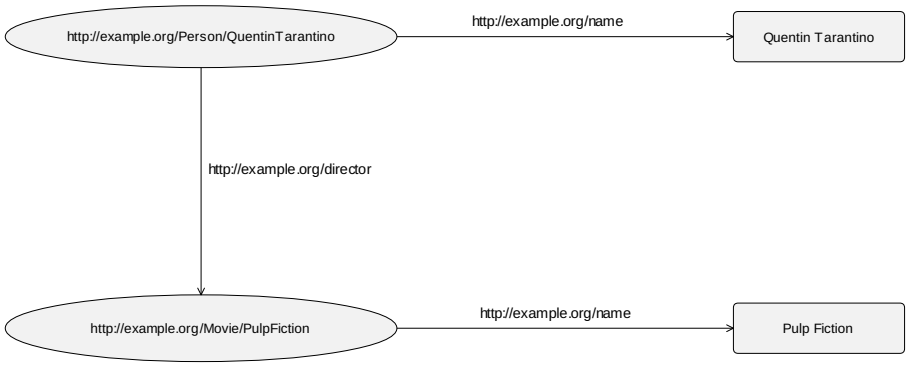

Figure 1 Simple example of graph-based triples representation.

The use of URIs in RDF is paramount. It is necessary to assign unique identifiers to each of the nodes so that they can be referred consistently across all the triples that describe the relationship. In a single dataset it is possible to use sequential numbers or strings to uniquely identify nodes. But for applications with multiple datasets, from heterogeneous sources, URI (especially URLs, where domain names are actually owned by the data publisher) ensure unique and consistent identifiers.

It is possible to provide information about a literal datatype. RDF supports the use of user defined and XML Schema types, which predefines a large range of basic types, including Boolean, integer, time and date. For typing complex concepts, such as resources and properties, one must use RDF Schema, as explain in subsection 2.2.

2.1.2 Persistence

Data in the RDF data model is persisted in a triplestore, a especial database for the storage and retrieval of triples. While relational databases are schema oriented, RDF triplestores are data oriented. That is, in relational databases data complies with a predefined schema, has explicit indexing and queries performs better with one-to-many relationships; on the other hand, data in a triplestore is semi-structured, triples are indexed and all relationships are many-to-many. Triplestores are built either as database engines from scratch or on top of existing relational database engines.

Just as SQL provides a query language across the relational database systems, SPARQL (Simple Protocol and RDF Query Language) provides a declarative interface for interacting with RDF graphs. It is an official W3C recommendation. SPARQL is both a standard query language and a data access protocol.

The SPARQL language consists of triple patterns, conjunctions (logical "and") and disjunctions (logical "or"). As in most the declarative languages, a query specifies a pattern in the data graph and the result set contains fragments that matched it.

2.1.3 Serialization

The RDF data model is very simple. Still, there are several methods available for serialization of RDF. A popular format is RDF/XML. In addition W3C introduced Notation 3 (N3), a text based format, that is related to Turtle and N-Triples. The non-XML serialization is easy to write by hand and, in some cases, easier to follow.

N-Triple notation is a very simple serialization, but still verbose. This simplicity makes this kind of serialization useful when hand-crafting datasets. Each line of output in N-Triple format represents a single statement containing a subject, predicate and object, followed by a dot. Every element is expressed as absolute URIs enclosed in angle brackets. The N-Triple simplicity causes redundant information that takes additional time to transmit and parse. While working with a small dataset it is not a problem, but the additional and redundant information becomes a liability when working with large amounts of data.

N3 condenses much of the information repetition in the N-Triple format. Every connection between nodes represents a triple. Since each node can have a large number of relationships, the number of characters can be reduced using prefixes. Similar to XML namespaces, N3 allows the definition of a URI prefix and identify resource URIs relative to a set of prefixes previously declared. N3 also reduces the repetition by allowing the combination of multiple statements about the same subject, by using a semicolon.

Turtle is a more verbose subset of N3 and an extension of N-Triples. It is a simple format for learning and making simple RDF Documents. The Turtle document is a collection of RDF-triples with `<subject> <relationship> <object>.` format. Each statement ends with a period and each element is an URI (except the `<object>` which can be a literal). If a subject has more than one statement, with different relationships, Turtle combines the multiple statements, using a semicolon. With Turtle it is also possible to define namespace prefixes, simplifying the document.

The RDF/XML is another way to serialize the RDF data model. Sometimes it is criticized for being difficult to read. Still it is one of the most frequently used formats. 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 subject (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 element. As with other XML documents, the top-level element

is used frequently to define other XML namespaces used through the document. Paths are always described starting with a graph node, using the URI reference for the node. Predicate links are specified as child elements of the node. Literal objects can be specified as the text of an element. And if the object is a node, a new element is created.

2.2 Resource Description Framework Schema

RDF provides a way to express simple statements about resources, using properties and values. However, users also need the ability to define vocabularies that they intend to use in those statements. In other words, users need to indicate that they are describing specific kinds or classes of resources and will use specific properties in that description.

Since RDF itself provides no means for defining classes and properties, it is used a RDF extension called RDF Schema (RDFS) [1, 6] to provide a type system for RDF. As in the type systems of some object-oriented programming languages, resources are instances of one or more classes, organized in a hierarchy.

A class in RDFS corresponds to the generic concept of a type or category. A class can be used to represent almost any category of things. A resource that belongs to a class is called its instance.

In RDF a class of resource is assigned with the `rdf:type` property whose value is the resource `rdfs:Class`. The relationship between two classes is described using the predefined `rdfs:subClassOf` property to relate these two classes. The meaning of this relationship is that any instance of the subclass is also an instance of the class. A class may be a subclass of more than one class. RDF Schema also defines all classes as subclasses of class `rdfs:Resource` since the instances belonging to all classes are resources.

Besides describing specific classes, users also need to be able to describe properties that characterize those classes. In RDF Schema all properties are described using the class `rdf:Property` and the properties `rdfs:domain, rdfs:range` and `rdfs:subPropertyOf` of RDFS.

The RDF Schema also provides a vocabulary for describing how properties and resources are related. The most important information is supplied by using the properties `rdfs:domain` and `rdfs:range`.

The `rdfs:domain` property indicates that a particular property applies to a designated class. In RDF, property descriptions are, by default, independent and have global scope. Then, a RDF Schema could describe a property without a domain being specified, being possible to extend the use of a property definition to a different situation.

The `rdfs:range` property indicates that the values of a particular property are instances of a designated 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.

RDF Schema provides a way to specialize properties as well as classes. The specialization relationship between two properties is described using the predefined `rdfs:subPropertyOf` property. A property may be subproperty of zero, one or more properties. The range and domain properties that apply to an RDF property also apply to each of its subproperties.

The Figure 2 represents the connection between a RDF and a RDF Schema. The blocks are properties, ellipses above the dashed line are classes and ellipses bellow the dashed line are instances. The RDF resources are related with the RDFS classes, by types. The RDFS relates classes hierarchically. The properties domain and range are constrained by classes.

In summary, RDF Schema provides schema information as additional descriptions of resources but does not determine how the descriptions should be used by an application. The

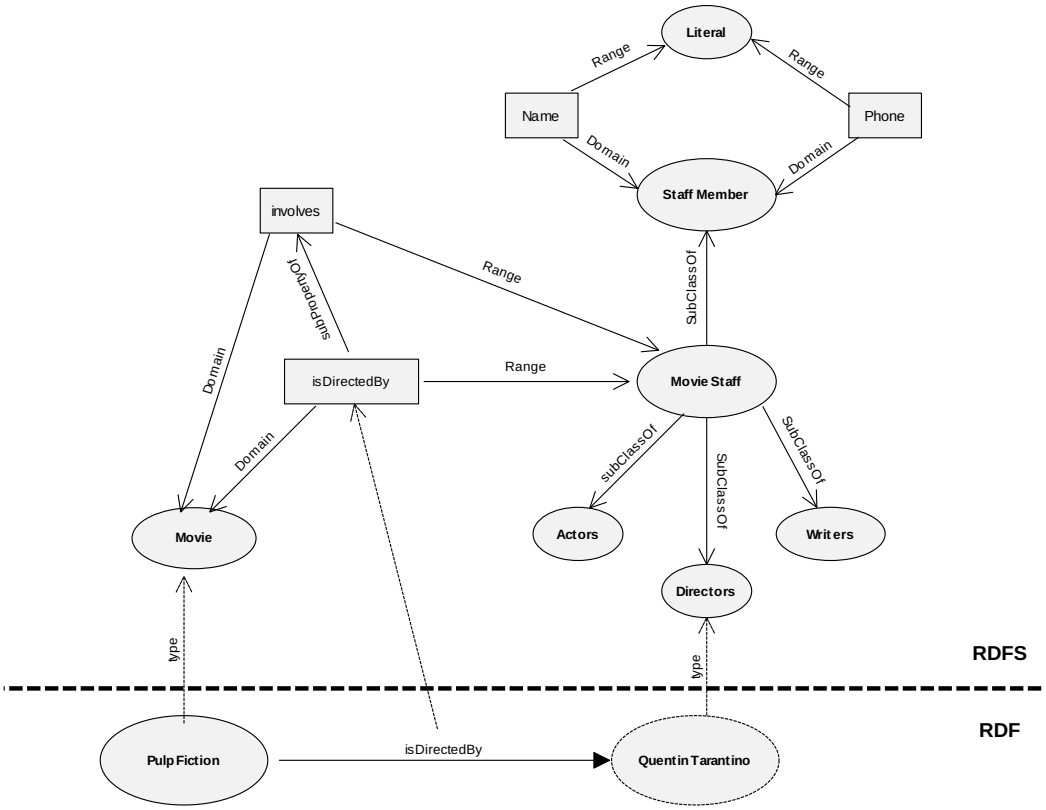

Figure 2 RDF and RDFS layers example.

statements in RDF Schema are always descriptions. They may also introduce constraints but only if the application interpreting those statements wants to treat them that way. All RDF Schemata provide a way of state additional information. If this information conflicts with the RDF data is up to the application to resolve it.

2.3 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. Some are complete database systems, as OpenLink Virtuoso, other are frameworks for developing semantic web applications, such as Jena, the framework selected for the implementation of DaPress.

OpenLink Virtuoso [8] is an open source edition of Virtuoso, an hybrid database management engine that supports multiple formats, including RDF, on top of a relational database.

The D2R Server [5] is another tool for publishing relational databases content as Linked Data. It supports RDF and HTML browsers to navigate the content of the database. It also has a SPARQL access endpoint.

Triplify [3] is a PHP plugin for web applications for small database contents (up to 100Mb). It exposes the semantic structure encoded in databases by making their content available as RDF, JSON or Linked Data. It is still a beta version and has not been updated since 2011.

Apache Jena [9] is an open source Semantic Web framework for Java. It provides an API to extract data from files, databases, URLs or a combination of these and produces RDF graphs that are serializable in RDF/XML, Turtle or N-Triple formats. This framework offers in-memory and persistent storage and supports SPARQL queries, among other query languages. Jena also provides a support for Web Ontology Language (OWL).

3 DaPress

DaPress is a tool that works as an intermediary between a semantic web client and a relational database, developed in Java with the Apache Jena Framework. This application extracts selected data from a relational database and transforms it into RDF. The generated triples are stored in a persistent triplestore using also a relational database[1].

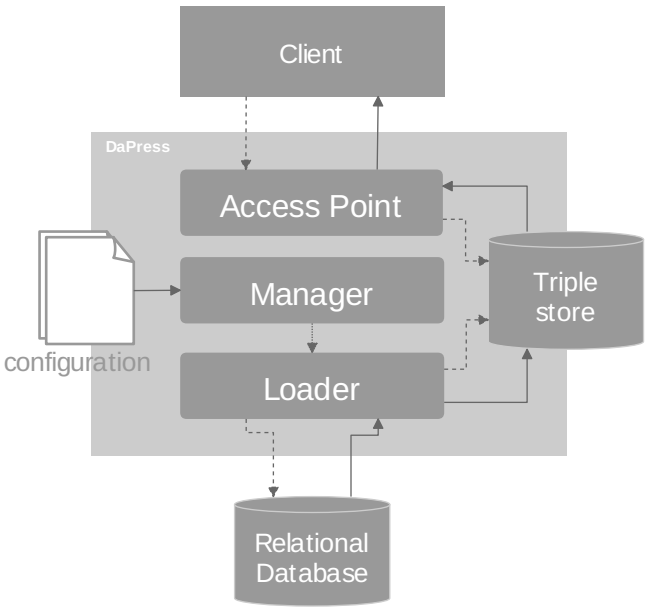

Figure 3 DaPress architecture.

As depicted in Figure 3 the architecture of DaPress aggregates three components:

Manager The module responsible for loading configurations, opening database connections, and controlling the other modules.

Loader Is in charge of converting relational data into the RDF and the RDFS, and store it in the triplestore using the mapping algorithms from Section 3.1.

Access Point Provides a SPARQL interrogation point. It is a specialized *servlet* providing a web service. For testing purposes, there is a simple web page to interact with the stored model, writing queries and viewing responses on a web browser, as shown on Figure 4.

[1] The source database and the triplestore may share the same database management system

Figure 3 illustrates how control (dashed arrows) and data (full arrows) flow through the system. Initially, the loading process is started by the Manager using the data in configuration files, in an operation that is periodically repeated. The Loader receives that information to execute queries to the external relational database. With the data the Loader creates the RDF and RDFS models and stores them in the triplestore. Later on, when a client makes a query to DaPress, the request is handled by the Access Point that interrogates the model stored in the triplestore.

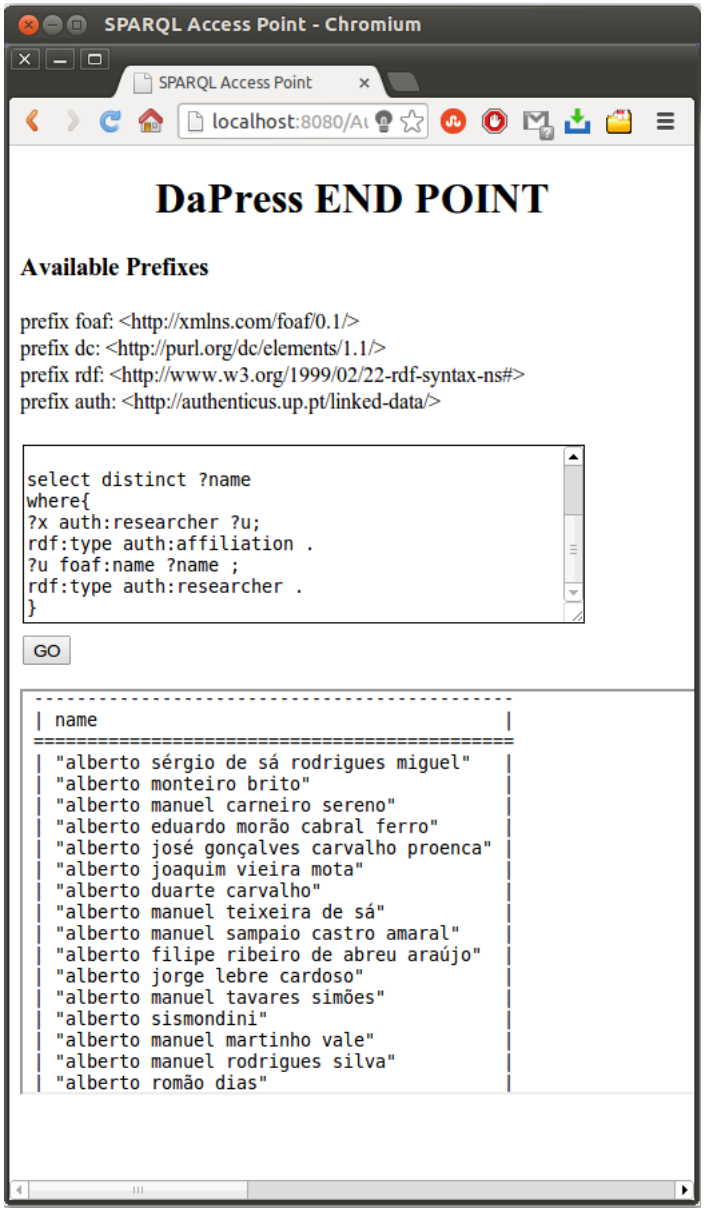

Figure 4 Screenshot of the DaPress access point web form.

The corner stone of DaPress is the mapping algorithm that converts relational data into RDF and RDF Schema driven by an XML configuration. The following two subsections detail both the algorithms and the DaPress configuration file.

3.1 Mapping Algorithm

This subsection presents the mapping algorithms of DaPress. For sake of clarity the algorithm for creating plain RDF triples from relational data is separated from the algorithm for creation RDF Schemata. Both algorithms use data provided by the XML data configuration file and data retrieved from the relational databases using SQL queries. Both algorithms produce a model, i.e. a collection of RDF triples. In DaPress these two models are merged in a single one and stored in the same triplestore.

In the following subsections, the RDF Mapping Algorithm and the RDFS Mapping Algorithm are described in detail. The last section presents an example of the application of the two algorithm.

3.1.1 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 Algorithm 1 the input is given by a collection of maps. Those maps resulting from configuration data have as prefix `selected`, such as `selectedTableNames`, returning a list of table names. In contrast, the input with prefix `get` corresponds to data coming from the relational database, such as: `getIds`, mapping table names to lists of ids; and `getValue`, mapping field and id pairs to values. Mappings with prefix `make` correspond to methods provided by the Jena API to create RDF elements, such as: `makeResource`, to make a resource from a type and an id.

The algorithm 1 iterates over the selected tables and for each one retrieves their identifiers from the database. For each identified record it creates a resource with `makeResource`. This resource is assigned with the *type* given by the `selectedResourceTypeName` map. This type is also assigned to the resource with the `type` property from the RDF vocabulary. For each field of the current record, a property is created using `makeProperty` function. The type associated with this property is given by the `selectedPropertyTypeName` map. According to the range selected for the field is created either a literal (typically a string) or a resource that is assigned to the object. In this last case the field value can be taken as a type, giving origin to a class hierarchy, as explain in the next sub-subsection. Finally a new statement is created and added to the model. The statement is created with the `makeStatement` using the previously created `subject`, `property` and `object`.

3.1.2 RDF Schema Mapping Algorithm

The algorithm for creating RDF Schema presented in Algorithm 2 is similar to the presented in the previous sub-subsection. Instead of creating RDF triples it creates classes and properties using the API available for that purpose in Jena. Although these methods create also RDF triples, they can be configured to use different vocabularies, such as RDF Schema or OWL, with the same implementation. To highlight the fact the model produced by this algorithm contains an ontology it is labelled as `ontModel`.

The RDF Schema algorithm has also the same inputs of the RDF algorithm. Although most of the data to create classes and properties comes from the configuration, 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 table is created a new ontology class with the type name assigned to that table. This new class is added to the model and is used as domain of the properties related to this type.

Algorithm 1: RDF Mapping algorithm.

```
Input  : selectedTableNames(), selectedFieldNames()
Input  : selectedResourceTypeName(), selectedPropertyTypeName()
Input  : selectedValueAsType(), selectedRangeTypeName()
Input  : getIds(), getValue()
Output : model
model ← ∅
for tableName ∈ selectedTableNames() do
    for id ∈ getIds(tableName) do
        type ← selectedResourceTypeName(tableName)
        for fieldName ∈ selectedFieldNames(tableName) do
            predicate ← makeProperty(selectedPropertyTypeName(fieldName))
            value ← getValue(fieldName, id)
            range ← selectedRangeTypeName(fieldName)
            if range = NULL then
                object ← makeLiteral(value)
            else
                if selectedValueAsType(fieldName) then
                    type ← value
                object ← makeResource(range, value)
            subject ← makeResource(type, id)
            model ∋ makeStatement(subject, predicate, object)
```

The selected fields of the current table are iterated and a property is created for each one. The previously created domain is immediately assigned to this property. The property range depends on the selected range for this field. If none was selected then it is a literal. This property is then added to the ontological model.

There is a special case when a field was selected as holding subclass names. In this case the records of this table must be iterated and a new ontological class created as subclass of the current domain. This new subclass is also added to the ontological model.

3.1.3 RDF and RDFS Mapping Algorithms Example

The following example illustrates how the two algorithms manipulate the information available in the relational database and the resulting RDF graph. Both tables, `Person` and `Town`, have an one-to-many relationship.

For each row in each table is generated a node. That node is identified by an unique URI and it is the subject of the triples. A node can be connected to another node. In the example, the property `isFrom` connects a person to a town. The node can also be connected to literal values such as the property `countryOf` that connects a town to its country.

Algorithm 2: RDF Schema Mapping algorithm.

```
Input  : selectedTableNames(), selectedFieldNames()
Input  : selectedResourceTypeName(), selectedPropertyTypeName()
Input  : selectedValueAsType(), selectedRangeTypeName()
Input  : getIds(), getValue()
Output : ontModel

ontModel ← ∅

for tableName ∈ selectedTableNames() do
    domain ← makeOntClass(selectedResourceTypeName(tableName))
    ontModel ∋ domain
    for fieldName ∈ selectedFieldNames(tableName) do
        property ← makeOntProperty(selectedPropertyTypeName(fieldName))
        makeDomain(property, domain)
        range ← selectedRangeTypeName(fieldName)
        if range = NULL then
            range ← literal
        makeRange(property, range)
        ontModel ∋ property
    if selectedValueAsType(fieldName) then
        for id ∈ getIds(tableName) do
            value ← getValue(fieldName, id)
            subClass ← makeOntClass(value)
            makeSubClass(subClass, domain)
            ontModel ∋ subClass
```

In the table `Person`, the gender column is used by the RDFS Mapping algorithm to create two different classes of `Person`, male and female. The prefix `a` is used to replace the full namespace URI `http://www.example.com`.

3.2 Configuration Files

The DaPress configuration is provided by an XML document that contains all the information required by the application. The document has four kinds of information: parameters to establish relational database connections; general configuration such as the SDB description file path and the delay of the updates; the selected resources and properties and related data. The structure of this document is formalized by an XML schema whose structure is depicted in diagram of Figure 6.

The most relevant part of the XML file is about the resources. The element `Resources` contains a sequence of elements for each resource. Each has a group of attributes that defines the name of the resource, the namespace, the type and the table in the relational database. The table is used in the SQL queries.

Each resource contains a set of properties. These properties also have a group of attributes defining the name of the property, the namespace, the column in the database, the range if applied, and a mandatory attribute. The mandatory attribute is a Boolean that allows the application to know if that attribute needs to exist; if True a `where` clause is added to the query stating that the current property (column in the query) must be `Not Null`.

Table: Person

id	name	gender	hometown
1	Maria	female	Porto
2	Helen	female	Chicago
3	Ian	male	Glasgow
4	Cristian	male	Madrid
5	Paul	male	London
6	Ana	female	Lisboa

Table: Town

id	name	country
1	Chicago	U.S.A
2	Glasgow	U.K.
3	Madrid	Spain
4	London	U.K.
5	Porto	Portugal
6	Lisboa	Portugal

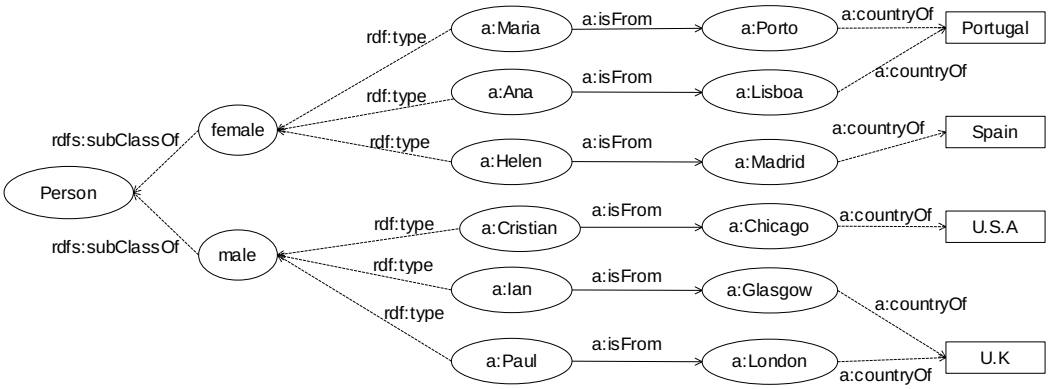

Figure 5 Example of the algorithm application.

A secondary configuration file is the SDB Description File. In this file is configured the connection to the triplestore an its path is defined in the DaPress main configuration file.

4 Validation

The validation of DaPress is based on the experience gained while publishing an existing relational database. The relational database selected for this purpose is part of *Authenticus* [7], 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.

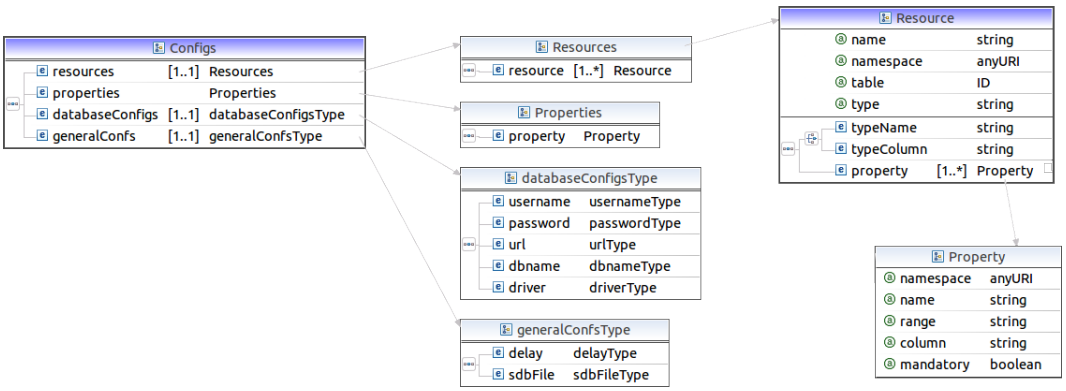

Figure 6 XML Configuration Schema.

The Authenticus database has currently 67 tables and a 210Mb of data. Of these 13 tables where selected with a size of 36,2Mb. The data sizes were computed from the SQL dumps of the referred sets of tables. The tables currently being used contain data on researchers, institutions, publications and journals. Some of the tables contain many-to-many relationships among these base entities. Tables from the original database that just support the web application where excluded from this mapping, such as those related to user management or containing precomputed values to speedup frequently requested listings.

The triplestore resulting from the mapping has a size of 153Mb when exported as an SQL dump. The significant increase in size is easily explained by the "explosion" in the number of records stored in the triplestore. The triplestore of DaPress and the database of Authenticus are currently on a two different relational database management system, although both running MySQL.

The machine where the mapping was processed is a Pentium 4 running at 2,4Ghz with 8Gb of RAM. It is operated by Linux Mandriva 2009 with a 2.6.19 kernel. The DaPress executed the mapping process in 194 minutes generating 1.456.353 triples, generating on average 1 triple in 0,006 seconds and producing 12Kb of data in one second. It should be noted that the machine available for these tests is rather old, with a single processor, thus the mapping should be even faster on a multi-core machine. Nevertheless, the order of magnitude of this time requires an incremental algorithm that is already planned for the next version of DaPress.

5 Conclusions and Future Work

This paper presents ongoing research to create a tool for publishing the content of relational database as linked data. The major contribution of this research is a pair of mapping algorithms, driven by configuration data stored in a single XML document, that convert relational data to RDF, and relational schemata to RDF Schema. These ideas are incorporated in the DaPress system and the design and implementation of this tool are also relevant contributions of this research. To validate the proposed approach DaPress was used with the content of the Authenticus database. Authenticus is a system that automatically assigns publication authors to known researchers and institutions. The experience gained using DaPress with Authenticus led to the identification of a number of issues in the current version that will be tackled in a near future.

The use of XML documents proved to be a simple and expedite way to define and store mapping information. However, it requires some knowledge of XML and the use of another tool to browse the relational database schema. An administrative web interface could show the tables and fields available on the relational database, enabling their selection and renaming for the mapping process.

The conversion from relational data to RDF does not increase the size of the data. However, the time necessary to convert the data, about a minute per megabyte, is too high for a regular update of the triplestore. The next version of DaPress must have an incremental algorithm to avoid reconverting unchanged data. This will be a challenge since DaPress does not assume any particular configuration of relational tables, such as the existence of time stamp fields with the creation/modification date. The current version of DaPress produces an ontology using RDF Schema, which includes the class hierarchy and the definition of properties based on those classes. This ontological data could be extended with OWL definitions, stating properties that cannot be represented in RDF Schema or that cannot be inferred from the relational schema. The use of OWL in DaPress must be investigated and

may be included in future versions.

Linked data published by DaPress 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 can be interlinked with the Digital Bibliography & Library Project (DBLP), a related system that stores publication data that also has a SPARQL access point.

After interlinking data in DaPress with related sources it will be possible to revert the publishing process from those sources into the local triplestore. That is, related RDF data from remote sources will be available for download into the triplestore of DaPress and translated to a relational database. For instance, RDF data from DBLP could be downloaded to the DaPress triplestore containing Authenticus data. The mapping configuration could then be used in the other direction, to convert data from the triplestore to the original relational database, avoiding the use of ad-hoc data converters.

References

1. *A Semantic Web Primer*. MIT Press, 2004.
2. *Programming the Semantic Web*. O'Reilly Media, 2009.
3. Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify: light-weight linked data publication from relational databases. In *Proceedings of the 18th international conference on World wide web*. ACM, 2009.
4. Tim Berners-Lee. Design issues: Linked data, 2006.
5. Christian Bizer and Richard Cyganiak. D2r server – publishing relational databases on the semantic web. Poster at the 5th International Semantic Web Conference, 2010.
6. Dan Brickey and R. V. Guha. Rdf vocabulary description language 1.0: Rdf schema. Technical report, World Wide Web Consortium, 2004.
7. Sylwia Teresa Bugla. Name identification in scientific publications. Master's thesis, Universidade do Porto, 2009.
8. Orri Erling and Ivan Mikhailov. Virtuoso: Rdf support in a native rdbms. In *Semantic Web Information Management*, pages 501–519. Springer, 2010.
9. Apache Software Foundation. Apache jena.
10. Graham Klyne and Jeremy J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. Technical report, World Wide Web Consortium, 2004.