

RDB2RDF: A relational to RDF plug-in for Eclipse

Edgard Marx¹, Percy Salas¹, Karin Breitman¹, José Viterbo^{2,*},[†] and Marco Antonio Casanova¹

¹*Informatics Department, Pontifícia Universidade Católica do Rio de Janeiro, Rua Marquês de S. Vicente, 225, Rio de Janeiro, Brazil - 22451-900*

²*Computer Science Department, Universidade Federal Fluminense, Rua Passo da Pátria 156 - Bloco E - 3º andar, Niterói, Brazil - 24210-240*

SUMMARY

The process of transforming data stored in relational databases (RDBs) into sets of Resource Description Framework (RDF) triples is known as triplification or RDB2RDF. It consists of two consecutive operations, schema extraction and data conversion. Schema extraction is a process similar to creating an external schema, and contains a collection of database views. The data conversion is divided into two steps. The first step consists of deciding how to represent database schema concepts in terms of RDF classes and properties, defining an RDB2RDF mapping. The second step consists of the actual conversion of relational data to RDF data instances, based on the mapping previously defined. Although the schema extraction is very well understood, the data conversion operation is still murky. Indeed, the World Wide Web Consortium RDB2RDF Working Group has been working to define a standard language, called R2RML, to describe RDB2RDF mapping files. The definition of the R2RML, however, is still undergoing changes. In this paper, we introduce an Eclipse plug-in that supports the entire conversion process. Its architecture takes into consideration the specificities of the triplification process by providing a modular structure that encapsulates the stable and well-understood components separately from the volatile, change-prone mapping strategies. The latter are accessible via a well-defined interface to promote information hiding and separation of concerns and to facilitate evolution. Copyright © 2012 John Wiley & Sons, Ltd.

Received 24 September 2011; Revised 7 May 2012; Accepted 3 July 2012

KEY WORDS: RDB2RDF; Linked Data; Semantic Web

1. INTRODUCTION

In the last few years, the Semantic Web has shown significant growth, as industry, government, and academia are gradually adopting appropriate standards [1]. Best Buy and Data.gov are well-known business cases in which the adoption of semantic standards, in particular, the Resource Description Framework (RDF) format [2], was fundamental to increase relevance and facilitate data interoperability [3, 4]. Nevertheless, if one compares the growth rates of the Web with those of the Semantic Web, there is still a significant gap. The main reason is that most of the existing Web sites, over 70%, derive their data from relational databases (RDBs), according to the Association for Computing Machinery [5].

Given the astounding amount of data stored in RDBs, a critical requirement for the evolution of the Semantic Web is the ability to convert data to Semantic Web-compatible formats, such as RDF and Web Ontology Language [6]. There are several strategies for converting relational data to RDF [7, 8], as well as a group of tools that claim automatic conversion. Most of the tools, however, implement proprietary mapping strategies to represent RDB schema concepts in terms of RDF classes and properties.

*Correspondence to: José Viterbo, Informatics Department, Pontifícia Universidade Católica do Rio de Janeiro, Rua Marquês de S. Vicente, 225, Rio de Janeiro, Brazil - 22451-900.

[†]E-mail: viterbo@ic.uff.br

The coexistence of competing mapping strategies and implementations led to an initiative to establish a common standard by which to coordinate efforts and promote overall interoperability. Led by the World Wide Web Consortium (W3C) RDB2RDF Working Group, researchers are currently working on a standard language to express mappings from RDBs into RDF, called the RDB to RDF Mapping Language (R2RML) [9]. A standardized mapping between RDB and RDF will allow the use of a single mapping specification. This feature will allow vendors to compete on functionality and features, rather than forcing database administrators to rewrite their entire relational data to a specific RDF mapping when they want to migrate their data from one database to another.

Practice has demonstrated that the most noteworthy RDF conversion tools available today, for example, Triplify [10], Virtuoso RDF Views [11], and D2RQ [12], to a certain extent, present scalability, extensibility, and usability problems. We believe that some of these shortcomings can be alleviated by the adoption of a plug-in approach. Among the advantages, plug-in architectures provide support to fine-grained modularity, an important software quality aspect when one wants to encourage code reuse and facilitate the development of extensions [13, 14]. This last aspect is particularly relevant to the conversion of relational data to RDF because, although the RDF specification is unique, RDBs come in many flavors. Plug-in frameworks also help in handling complexity, simplify configuration and application deployment, and, more importantly, enable users to extend the functionality of existing applications with self-developed modules without having to modify the rest of the source code.

In this paper, we propose an extensible plug-in for Eclipse[‡] that supports the conversion of relational data to RDF. The remainder of this paper is organized as follows. In Section 2, we discuss the basic concepts. In Section 3, we summarize related work. In Section 4, we describe the plug-in architecture. In Section 5, we present a running example. Finally, in Section 6, we present concluding remarks and suggestions for future work.

2. BASIC CONCEPTS

2.1. Resource Description Framework

The RDF is a language for representing information about resources on the World Wide Web [2]. It was particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page. It can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved from the Web. Examples include information about items available from online shopping facilities (e.g., information about specifications, prices, and availability) or the description of users' preferences for information delivery.

Resource Description Framework does not replace the existing data models, such as the entity-relationship model. Furthermore, RDF was designed for situations where Web data need to be processed by applications, rather than being displayed for people. As such, RDF provides a common framework for expressing data on the Web so that it can be exchanged between software applications. The ability to exchange data between different applications means that the data may be made available to applications other than those for which they were originally intended. This fact makes RDF a good integration platform for data from multiple sources and has greatly contributed to placing RDF as the de facto standard for representing Linked Data on the Web [7].

To some extent, RDF is a lightweight ontology language to support interoperability between applications that exchange machine-understandable data on the Web. RDF has a very simple and flexible data model, based on the idea of making statements about resources in the form of (S, P, O) expressions, where

- S is an Internationalized Resource Identifier (IRI),[§] called the subject of the statement;
- P is an IRI, called the property (also called the predicate) of the statement, which denotes a binary relationship;

[‡]The plug-in is available for download at <http://lod2.inf.puc-rio.br/site/download/>

[§]<http://www.ietf.org/rfc/rfc3987.txt>

- O is either an IRI or a literal, called the object of the statement; if O is a literal, then O is also called the value of property P .

Those expressions are known as RDF triples. A directed graph is a convenient way to visualize triples. In such graph, subjects and objects are nodes, and a triple (S, P, O) is represented as an edge from the subject to the object, labeled as the predicate. As an example, in Figure 1, we show an RDF graph representing four triples, which describe properties about the subject ‘Rio de Janeiro’, identified as the IRI ‘http://dbpedia.org/resource/Rio_de_Janeiro’.

2.2. Transforming relational data to Resource Description Framework

The publication of data stored in RDBs in RDF is a critical step in the move to the Web of Data. Also known as RDB2RDF [15], the process of transforming data stored in RDBs to the RDF format is currently the focus of many research efforts [10–12]. Briefly, the process consists of two consecutive operations: schema extraction and data conversion. Schema extraction is a process similar to creating an external schema, containing a collection of database views. This step is necessary because not all information in the database is relevant to be published as RDF triples, for example, confidential or private information contained in a database should not be published.

The data conversion process consists of defining how to represent database schema concepts in terms of RDF classes and properties [16], followed by the conversion of relational data to triples. The first step in this process is creating a mapping file, which is a document containing a series of individual mappings that describe how the views in the external schema should be mapped into RDF triples. This process uses one or more RDF vocabularies, selected to describe schema components in terms of RDF concepts. The mapping file serves as input to the conversion step, which consists of the generation of the actual RDF triples from the data stored in the database. Figure 2 shows, on the right, the RDF triples resulting from the conversion of some RDB data, on the left.

A large number of strategies for mapping relational data to RDF can be found in the literature [10–12]. Sahoo *et al.* provided a comprehensive analysis of existing approaches, in which the authors analyzed over 15 strategies, classified into three broad classes: Proof of Concept Projects,



Figure 1. A Resource Description Framework graph describing ‘Rio de Janeiro’.

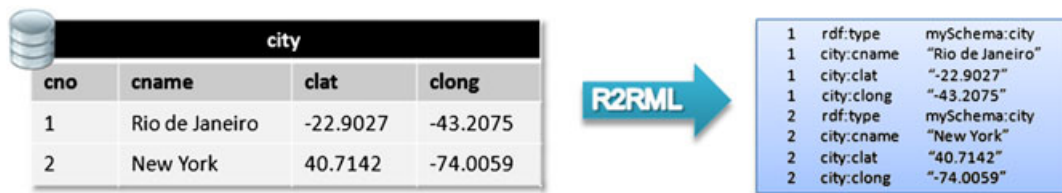


Figure 2. Relational database to Resource Description Framework example.

Table I. RDB2RDF conversion tools.

Tool	Interface	Automation	Mapping language	Extensible mapping
Triplify	Text editor	Manual	SQL	NO
D2RQ	Text editor	Manual/auto	RDF based	NO
Virtuoso RDF Views	Text editor	Manual	RDF based	NO
RDB2RDF plug-in	Graphical/text editor	Manual/auto	R2RML	Yes

Domain-specific Projects, and Tools/Applications. We refer the reader to their article for more details [7]. More important than the number of existing strategies is the fact that they are undergoing constant evolution. During the period that this manuscript was being prepared, W3C issued three versions of the working drafts of both the Direct Mapping language [17], which defines a straightforward transformation in which the RDF vocabulary directly reflects the names of database schema elements, and the R2RML, a language for expressing customized mappings from RDBs to RDF datasets [18–20].

The R2RML is regarded as the de facto standard for the conversion of relational data to RDF format. Every R2RML mapping is tailored to a specific database schema and one or more target vocabularies [9]. The input to an R2RML mapping is an RDB that conforms to that schema. The output is an RDF triple set, in which the RDF statements were created (triplified) using concepts from the target vocabularies. These mappings themselves are represented as an RDF graph and referred to as the mapping graph. When a mapping graph is encoded in the Turtle[¶] syntax, it becomes an R2RML mapping document.

3. RELATED WORK

Sahoo *et al.* [7] pointed out that researchers and practitioners have provided different mechanisms with which to tackle the RDB2RDF conversion process. However, most of the current RDB2RDF tools provide different proprietary mapping languages for the mapping process. The main features of the most relevant tools, which are Triplify, D2RQ, and Virtuoso RDF Views, are summarized in Table I, together with the RDB2RDF plug-in, and discussed later.

Even though these tools have very expressive mapping strategies, they do not follow the proposed standard (R2RML). Furthermore, some of them [10, 11] do not provide an automated method to perform the mapping, leaving all the work to the users, who, in most cases, are not familiar with the required technique. By contrast, our tool offers the possibility of generating mappings in R2RML and is extensible, which is of particular interest in order to allow users to create automatic mappings for other techniques that may arise.

3.1. Triplify

Auer *et al.* [10] described Triplify, a simplified approach based on mapping Hypertext Transfer Protocol requests containing Uniform Resource Identifiers onto RDB queries. Triplify motivates the need for a simple mapping solution through using SQL as the mapping language to transform database query results into RDF triples and Linked Data. The mapping is carried out manually in a text editor. It uses the table-to-class and column-to-predicate approach for transforming SQL query results into the RDF data model. This transformation process can be performed on demand through Hypertext Transfer Protocol or in advance. The approach promotes the reuse of mapping files, through a collection of configuration files for common relational schemata. It can be easily integrated and deployed with the numerous widely installed Web applications such as WordPress,

[¶]Terse RDF Triple Language—defines a textual syntax for RDF with levels of compatibility with the existing formats as triple pattern that allows an RDF graph to be completely written in compact and natural text form, with abbreviations for common usage patterns and datatypes.

Gallery, and Drupal. Triplify also includes a method for publishing update logs to enable incremental crawling of linked data sources. The approach was tested with 160 GB of geo data from the OpenStreetMap project and exhibited a high flexibility and scalability.

3.2. D2RQ

D2RQ [12] generates the mapping files automatically, using the table-to-class and column-to-predicate approach. D2RQ uses a declarative language, implemented as a Jena graph [21], to define the mapping file. The approach allows RDBs to offer their contents as virtual RDF graphs without replication of the relational data as RDF triples. The tool can also provide an RDF dump of the RDB, if required. In the virtual access mode, the mapping file is largely used for translating SPARQL to SQL queries. The mapping file may be customized by the user, thereby allowing the user to reuse standard ontology terms in the mapping process.

3.3. Virtuoso RDF Views

Virtuoso RDF Views [11] use the table-to-class approach for automatic generation of the mapping file. The mapping file, also called RDF view, is composed of several declarations called quad map patterns, which specify how the column values of tables are mapped to RDF triples. Similarly to D2RQ, Virtuoso supports mapping arbitrary collections of relational tables into RDF without having to convert the whole data into RDF triples. The data returned by the process are presented as virtual RDF graphs, without the physical creation of RDF datasets, according to the mapping file represented in quad map patterns. Also, the mapping file can be stored as triples and therefore can be queried via SPARQL.

4. ARCHITECTURE

The architecture of the plug-in was conceived taking into consideration the opposing nature of our requirements. Part of the solution requires a component able to communicate with RDBs and import metadata and the database schema. The requirements for this component are well understood and quite stable. The remaining part requires a flexible component to deal with the implementation of a variety of relational to RDF mapping algorithms that are volatile, as argued in the previous section.

Our solution was to split the implementation into two separate modules, as illustrated in Figure 3. The first module, the *Schema Generator*, is an Eclipse plug-in responsible for (1) communicating with the RDB (using public domain drivers), (2) providing a mechanism in which to build an external schema, which defines the data that will be converted to RDF, and (3) importing the selected elements from the database schema. The second module, the *Mapper*, implements one of a wide choices of mapping algorithms. This architecture separates the two main concerns of the proposed plug-in but, more importantly, promotes isolation of the mapping algorithm that is more likely to change. The intent is to provide a way in which to change/update the mapping algorithm without having to rewrite the plug-in.

As can be seen in Figure 4, our implementation is divided into four main packages: *Mapper*, *Schema*, *Preferences*, and *Handler*. The *Mapper* package contains classes that define the relational to RDF mapping, such as the *Mapper interface* and the *MapperPluginService* class. It automatically

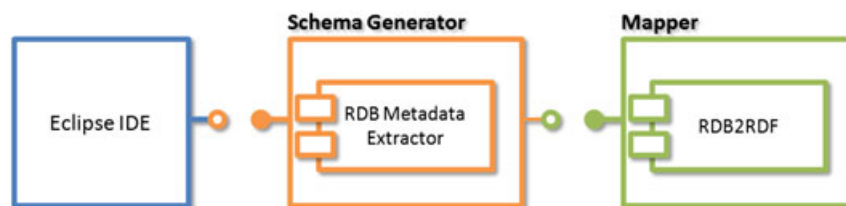


Figure 3. Plug-in architecture.

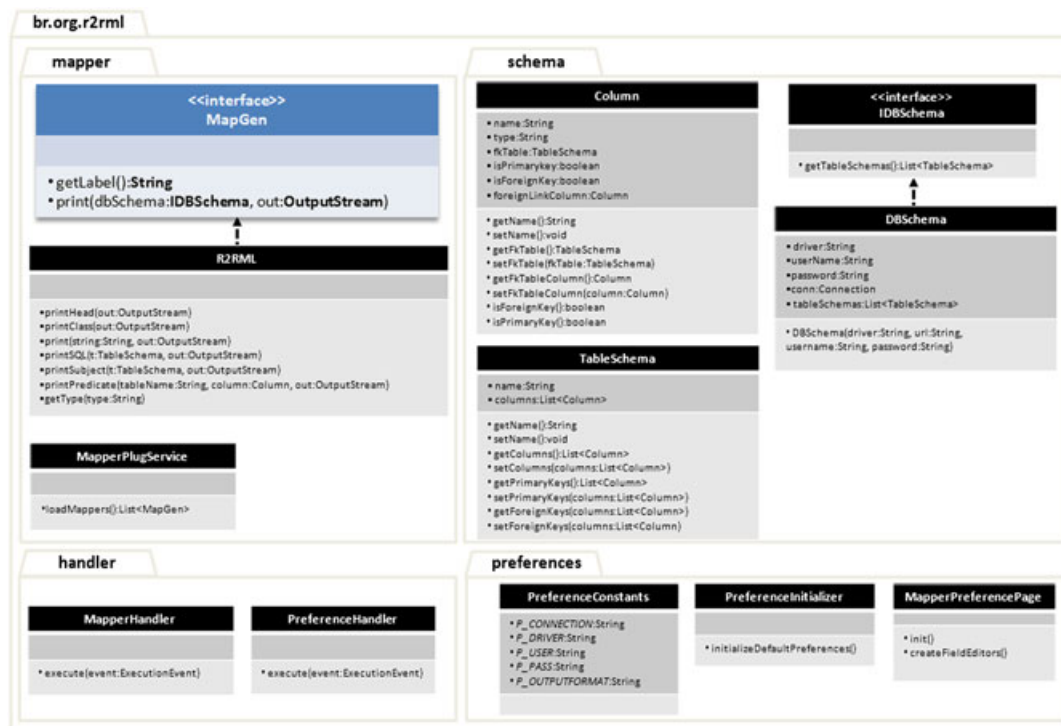


Figure 4. Class diagram of the mapper implementation.

identifies and loads all available *Mapper* implementations, such as the *R2RML* class that implements the R2RML mapping algorithm. The *Schema* package has model classes that will be used to represent the schema extracted from the RDB, which will be used to generate the mapping and will be passed as a parameter to the *Mapper* implementations through the *print* method. The other two packages contain external schema layer classes. The *Preferences* package has classes for the preferences window, where the user can, among other tasks, choose the desired mapping algorithm. The *Handler* package has handlers that trigger the execution of the selected algorithm (*MapperHandler*) or the preferences window (*PreferencesHandler*). Once the *MapperHandler* is triggered, it will extract the schema of the selected database and invoke the *MapperPluginService* to obtain the *Mapper* implementation selected in the preference window. Then the *print* method of the *Mapper* will be invoked, and the result will be placed in an output stream to be shown on the editor window.

4.1. Schema Generator

The database connection and metadata extraction are performed through the Java Database Connectivity (JDBC) driver, which is available for several different RDBs. Our current implementation includes drivers for the major commercial and open source databases, including SQL Server, Postgres, MySQL, and Oracle. Metadata extraction requires the proper installation of the database driver. It requires also that the users manually supply all the necessary connection parameters. In Figure 5, we show the plug-in configuration graphical user interface, where users must enter the required parameters, for example, driver names, server addresses (host), connection ports (port), user name, password, and database name.

The Schema Generator provides an interface in which users can select the database schema elements to be published in RDF. Apart from obvious confidentiality restrictions, one must identify which data is of interest. It is common practice in RDB design and implementation to store partial indexes, as well as internal codes, to improve performance. Although relevant for the operation of the RDB, they are meaningless to the conversion to RDF. Therefore, it is paramount to decide which database schema elements are of interest and should be retained in the conversion process.

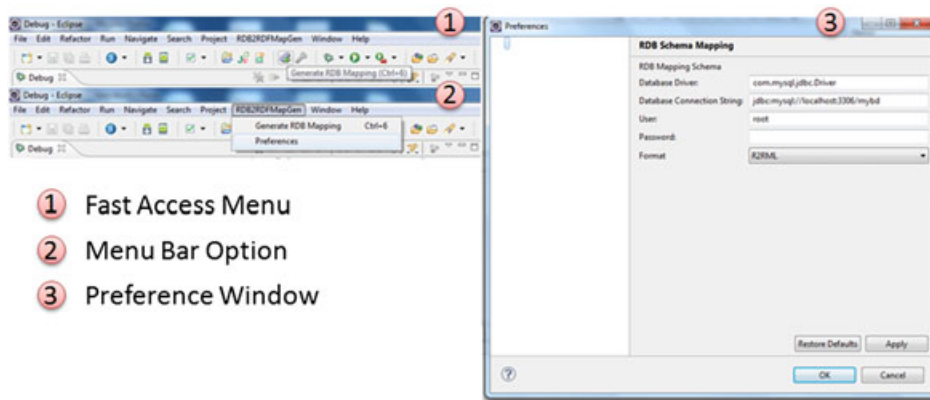


Figure 5. RDB2RDF plug-in graphical user interface.

This mechanism is similar to the creation of an external schema, containing a set of views over the relational schema. We recall that each view has a view scheme, specifying the view name and the view column names, and a view definition, defined by a query over the underlying relational schema. The mechanism therefore enables users to customize their mappings in that it allows users to choose the source data that will be triplified, leaving out sensitive, private, and nonrelevant data [22].

4.2. Mapper

As RDB2RDF mapping algorithms are in constant evolution, an implementation of such tools must be flexible and amenable to change. Inspired by the information hiding [23] principle, we isolated the implementation of the mapping algorithms in a separate component, accessible through an interface. In this way, we protect those parts of the implementation that are more stable from those that are more likely to undergo modification and eventual substitution. By doing so, we promote software reuse and facilitate evolution.

Maintenance or the addition of new mapping algorithms is possible because the plug-in is built on top of the service provider interface (SPI). SPI is a well-known API, widely adopted in the development of reusable components in several technologies, for example, JDBC, Java Cryptography Extension, Java Naming and Directory Interface, Java Application Program Interface for XML Processing, and Geographic Information Systems (Geotools). A description and a full specification of SPI can be found in [7, 15], respectively. The use of SPI makes the proposed plug-in change-ready, as it allows the update/addition of new algorithms by implementing a simple interface in which users can substitute (or modify) the current mapping algorithm, adding third-party libraries, or calling an external API, without having to modify the implementation of the rest of the plug-in.

Figure 4 shows the proposed implementation, highlighting the *MapGen* interface. The plug-in architecture is a relational-to-RDF conversion framework whose only hotspot is the *MapGen* interface, responsible for the adaptation to different mapping algorithms. This interface accepts any mapping algorithm implementation compatible with the format described in Listing 1 and exemplified by Figures 7 and 8. It implements two methods: *getLabel*, which returns the name or version of the mapping algorithm; and *print*, which receives a set of view definitions (the external schema), their metadata, and an output stream, where the results will be placed after the conversion process.

According to the SPI specification, services are composed of a well-known set of interfaces and (abstract) classes; new services are identified by a provider-configuration file located in the resource META-INF/services directory. Our approach uses this information to automatically detect when new mapping implementations are deposited in the class path. Whenever a new algorithm is added, the plug-in automatically adds it as a new option into the configuration window, as shown in Figure 5.

4.3. Mapping implementation: R2RML

We illustrate later how the plug-in implements the algorithm that supports the R2RML specification [19] using the direct mapping [17]. For the full specification of the R2RML mapping language, we refer the reader to its most recent release, available at [9].

The algorithm contains nine classes—*GraphMap*, *Join*, *LogicalTable*, *ObjectMap*, *PredicateMap*, *PredicateObjectMap*, *RefObjectMap*, *SubjectMap*, and *TriplesMap*—and few additional properties. The *TriplesMap* class is an abstraction of the mapping and contains a *LogicalTable*, a *SubjectMap*, and a *ForeignKeyMap* for each table of the database schema, and several *PredicateObjectMap*, one for each table column, as per the document issued by the W3C.

Although the algorithm is useful for automating the mapping process, the R2RML specification offers advanced techniques that allow a more comprehensive and powerful mapping specification than that automatically generated. For instance, it allows users to specify new datatypes to define literals, the language of the component object of a predicate–object pair (e.g., @en for English or @pt for Portuguese), and even to specify how to create the Uniform Resource Identifier of a specific subject. Our interpretation of this specification is summarized by the algorithm described in Listing 1.

Input: the external schema, containing a set of view schemes
Output: the R2RML direct mapping

```

1 for each view scheme in the external schema
2
3   create the LogicalTable map of the view scheme;
4
5   if there is primary key in the view scheme
6     create the SubjectMap for the primary key;
7
8   for each column in the view scheme
9     create the predicateObjectMap for the column;
10
11  if there is a foreign key in the view scheme
12    create the foreignKeyMaps;
13
14 print the R2RML mapping in a given output stream;
```

Listing 1: Mapping algorithm.

The algorithm illustrated in Listing 1 works on an external schema, containing a set of view schemes, defined over the RDB schema. It is responsible to generate mappings for each view scheme of the external schema. The process is quite straightforward and consists of three consecutive steps in which elements of the view schemes, for example, columns, primary and foreign keys, are used to compose the RDF triples, predicates, subjects and objects. The process is repeated for each view scheme of the external schema.

4.4. Empirical results

As we mentioned before, the plug-in architecture provides a modular structure that encapsulates stable and well-understood components separately from the volatile, change-prone mapping strategies. The components that implement mapping algorithms are accessible via a well-defined interface to promote information hiding and separation of concerns, thus facilitating evolution. In order to assess such features, we implemented three different Mapper algorithms: Direct Mapping and two versions of R2RML. The results of this experiment are summarized in Table II, in which column *Algorithm* lists the algorithm version implemented, column *Class* shows the classes changed or extended, and, finally, column *Time* shows the approximate time spent in the code development and test (excluding the time to read and learn the specification).

Table II. Time spent in implementing and testing Mapper algorithms.

Algorithm	Class	Time
Direct Mapping	DirectMapping	1 week
R2RML (October 10, 2010)	R2RML_R1	8 h
R2RML (March 24, 2011)	R2RML	3 h

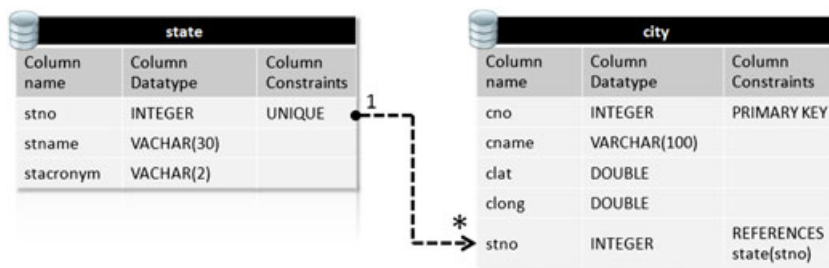


Figure 6. Input data example.

In each version, the users of the plug-in had only to create a single class containing the algorithm and implementing the *MapGen* interface. In the first algorithm, the Mapper module generates the triples by using the Direct Mapping technique, and, as such, the class *DirectMapping* had to be implemented and tested, what took about 1 week. The second algorithm, which implements the first revision of R2RML (from October 10, 2010), was developed and tested in approximately 8 h. We believe that this reduced time exemplifies the huge time saving in the development of a single class, whereas the rest of the plug-in keeps unchanged. The last algorithm implements a new version of R2RML (from March 24, 2011) and was created by introducing minor modifications in the second, what, in part, justifies the little time spent in its implementation, approximately 3 h. All the Mapper versions were tested by comparing the mapping generated with the expected conforming the specification of R2RML and Direct Mapping.

5. RUNNING EXAMPLE

As we described in the previous section, the input to our R2RML mapping component is an RDB that conforms to the RDB schema and JDBC standard, whereas the output is an R2RML mapping file. The mapping is itself also represented as an RDF graph, that is, in this case, RDF is not only the target data model but also the formalism in which the R2RML mapping is represented.

In the following example, we explain how the input data is used to generate the respective R2RML mapping file. Let us assume that the input to the mapping process is the relational schema represented in Figure 6, which consists of two tables, State and City, each with a single column as primary key and with one foreign key between them.

The output generated for this input corresponds to the output R2RML files shown in Figure 7 for the State and City tables.

To illustrate the flexibility and ease of use of the R2RML file obtained as output, we modified the files to replace the terms generated automatically by terms from GeoNames¹¹ vocabulary. Figure 8 shows two listings with the modified files. Observe that some of the default terms used in the State and City table mappings were replaced by terms from the GeoNames

¹¹GeoNames is a geographical database available for download at www.geonames.org under a Creative Commons Attribution License. GeoNames contains over 10 million geographical names and 7.5 million unique features. The entire GeoNames vocabulary is covered by the GeoNames Ontology.

<pre> @prefix rr: <http://www.w3c.org/ns/r2rml#> . @prefix ex: <http://www.example.org/#> . <#TriplesMap1> a rr:TriplesMap; rr:logicalTable [rr:sqlQuery "" Select stname , stacronym , stno from state ""]; rr:subjectMap [rr:column "stno"; rr:class ex:state;]; rr:predicateObjectMap [rr:predicateMap [rr:predicate state:stname]; rr:objectMap [rr:column "stname"];]; rr:predicateObjectMap [rr:predicateMap [rr:predicate state:stacronym]; rr:objectMap [rr:column "stacronym"];]; </pre>	<pre> @prefix rr: <http://www.w3c.org/ns/r2rml#> . @prefix ex: <http://www.example.org/#> . <#TriplesMap2> a rr:TriplesMap; rr:logicalTable [rr:sqlQuery "" Select cno , cname , clat , clon , stno from city ""]; rr:subjectMap [rr:column "cno"; rr:class ex:city;]; rr:predicateObjectMap [rr:predicateMap [rr:predicate city:cname]; rr:objectMap [rr:column "cname"];]; rr:predicateObjectMap [rr:predicateMap [rr:predicate city:clat]; rr:objectMap [rr:column "clat"];]; rr:predicateObjectMap [rr:predicateMap [rr:predicate city:clon]; rr:objectMap [rr:column "clon"];]; rr:predicateObjectMap [rr:predicateMap [rr:predicate city:stno]; rr:parentTriplesMap ex:state; rr:joinCondition "(child.)stno = (parent.)stno";]; </pre>
---	---

Figure 7. R2RML mapping files for City (right) and State (left).

<pre> @prefix rr: <http://www.w3c.org/ns/r2rml#> . @prefix gn: <http://www.geonames.org/ontology#> . <#TriplesMap1> a rr:TriplesMap; rr:logicalTable [rr:sqlQuery "" Select stname , stacronym , stno from state ""]; rr:subjectMap [rr:column "stno", rr:class gn:A;]; rr:predicateObjectMap [rr:predicateMap [rr:predicate gn:name]; rr:objectMap [rr:column "stname"];]; rr:predicateObjectMap [rr:predicateMap [rr:predicate gn:shortName]; rr:objectMap [rr:column "stacronym"];]; rr:predicateObjectMap [rr:predicateMap [rr:predicate gn:countryCode]; rr:objectMap [rr:object "BR"];]; </pre>	<pre> @prefix rr: <http://www.w3c.org/ns/r2rml#> . @prefix gn: <http://www.geonames.org/ontology#> . <#TriplesMap2> a rr:TriplesMap; rr:logicalTable [rr:sqlQuery "" Select cno , cname , clat , clon , stno from city ""]; rr:subjectMap [rr:column "cno"; rr:class gn:P;]; rr:predicateObjectMap [rr:predicateMap [rr:predicate gn:name]; rr:objectMap [rr:column "cname"];]; rr:predicateObjectMap [rr:predicateMap [rr:predicate gn:lat]; rr:objectMap [rr:column "clat"];]; rr:predicateObjectMap [rr:predicateMap [rr:predicate gn:lon]; rr:objectMap [rr:column "clon"];]; rr:predicateObjectMap [rr:predicateMap [rr:predicate gn:parentFeature]; rr:parentTriplesMap gn:A; rr:joinCondition "(child.)stno = (parent.)stno";]; </pre>
--	---

Figure 8. R2RML mapping files for City (right) and State (left) adopting the GeoNames vocabulary.

vocabulary. Moreover, the relationship between these two classes, which was previously established by the predicate *city:stno*, was changed to the predicate *gn:parentFeature*. These changes facilitate the consumption of the triples by applications that recognize the GeoNames vocabulary.

Figure 9 shows an example of a mashup created using the modified mappings that adopt the Geonames vocabulary. In the example, the markers indicate large Brazilian cities classified as ‘Geoname P’ with the Geonames vocabulary.



Figure 9. Example of a mashup using data obtained from a relational database using R2RML. In the example, the markers are large Brazilian cities classified as 'Geoname P'.

6. CONCLUSION

In this paper, we introduced an extensible Eclipse plug-in that converts relational data to RDF. On one hand, as the proposed standard for the RDB2RDF conversion process is still unstable, the proposed architecture should be extensible and welcoming to change. On the other hand, a large part of the required functionality requires database communication and the construction of a set of views, for which there is a solid body of knowledge and a roster of consolidated techniques. Our architecture builds on this dichotomy to leverage its implementation. We followed a plug-in approach that accounts for the separation of the well-understood and less likely to change components from the component that implements the mapping algorithm, which requires more frequent revisions and potential customization. Access to this component is granted through a standard interface to facilitate its use by nonexpert users. Our experiment described in Section 4.4 also demonstrated that the adoption of a layered architecture, oriented as a plug-in, enabled the abstraction of knowledge and the reuse of components, reducing the development time of new plug-in components (Mappers).

The RDB2RDF plug-in is available under the Academic Free License** and is currently being adopted by at least one company specialized in semantically rich systems,†† which provided

**<http://opensource.org/licenses/academic.php>

††VISTology Inc. (<http://www.vistology.com>)

positive feedback about it. This fact partially justifies our belief that the development of plug-ins such as the one proposed here represent an important advance in the popularization of Semantic Web tools. However, there is still much to be done, especially if we focus on nonexpert adoption. The most critical issues, in our opinion, are the following:

- Facilitate data selection: Not all relational data need always be mapped to RDF. Advanced user interfaces, which help identify sensitive data and exclude it from the process, are needed.
- Choice of vocabulary: It is still important to encourage the creation of algorithms for automatic vocabulary matching, thus reducing the overhead and need for manual intervention in the process.
- Provide support for relational datasets in the cloud: This issue is of particular interest with respect to Open Government Data and data stored in open access repositories, such as the Azure Marketplace.

ACKNOWLEDGEMENTS

This research was made possible by grants E-26/170028/2008 from FAPERJ and 557.128/2009-9 from CNPq, and was carried out with the Brazilian Web Science Institute.

REFERENCES

1. Breitman KK, Casanova MA, Truszkowski W. *Semantic Web: Concepts, Technologies and Applications*, Vol. 1. Springer: Londres, 2006. 337.
2. Manola F, Miller E. RDF primer, W3C recommendation, 2004. Available from: <http://www.w3.org/TR/rdf-primer/> [last accessed August 11, 2011].
3. Brinker S. Best Buy jump starts data Web marketing. Available from: <http://www.chiefmartec.com/2009/12/best-buy-jump-starts-data-web-marketing.html> [last accessed December 13, 2010].
4. Ding L, Difranzo D, Graves A, Michaelis JR, Li X, McGuinness DL, Hendler J. Data-gov Wiki: towards linking government data. In *AAAI Spring Symposium on Linked Data Meets Artificial Intelligence*. AAAI, Palo Alto: California, 2010.
5. He B, Patel M, Zhang Z, Chang KC. Accessing the deep Web. *Communications of the ACM* 2007; **50**(5):94–101.
6. Web Ontology Language. W3C Recommendation 10 February 2004. Available from: <http://www.w3.org/2004/OWL/> [last accessed August 11, 2011].
7. Sahoo SS, Halb W, Hellmann S, Idehen K, Thibodeau Jr T, Auer S, Sequeda J, Ezzat A. A survey of current approaches for mapping of relational databases to RDF. *W3C RDB2RDF Incubator Group Report*, 2009. Available from: http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf [last accessed August 11, 2011].
8. Salas PE, Marx E, Mera A, Viterbo J. RDB2RDF plugin: relational databases to RDF plugin for Eclipse. In *Proceeding of the 1st Workshop on Developing Tools as Plug-ins (TOPI '11)*. ACM: New York, NY, USA, 2011; 28–31.
9. Das S, Sundara S, Cyganiak R (eds). R2RML: RDB to RDF Mapping Language. W3C RDB2RDF Working Group. Available from: <http://www.w3.org/TR/r2rml/> [last accessed December 2010].
10. Auer S, Dietzold S, Lehmann J, Hellmann S, Aumueller D. Triplify: lightweight linked data publication from relational databases. In *Proceedings of the 18th International Conference on World Wide Web*. ACM: Madrid, Spain, 2009; 621–630.
11. Erling O, Mikhailov I. RDF support in the Virtuoso DBMS. In *Proceedings of the 1st Conference on Social Semantic Web, volume P-113 of GI-Edition—Lecture Notes in Informatics*, Vol. 113. Bonner Kollen Verlag: Leipzig, Germany, GI, September 2007; 59–68.
12. Bizer C, Seaborne A. D2RQ—treating non-RDF databases as virtual RDF graphs. In *3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, November 2004.
13. Cox B. Planning the software industrial revolution. *IEEE Software* November 1990; **7**(6):25–33.
14. McIlroy MD. Mass produced software components. In *Software Engineering: Report on a conference by the NATO Science Committee (Garmisch, Germany, Oct.)*, Naur P, Randell B (eds). NATO Scientific Affairs Division: Brussels, Belgium, 1968; 138–150.
15. Prud'hommeaux E, Hausenblas M. Use cases and requirements for mapping relational databases to RDF, 2010. (Available from: <http://www.w3.org/TR/rdb2rdf-ucr/>) [last accessed August 11, 2011].
16. Salas P, Breitman K, Viterbo J, Casanova MA. “Interoperability by design using the Std-Trip Tool: an a priori approach” (Triplification Challenge Submission). In *Proceedings of the 6th International Conference on Semantic Systems 2010 (ISEMANICS-10)*. ACM: New York, NY, USA, 2010. Article No. 43.
17. Arenas M, Prud'hommeaux E, Sequeda J (eds). A direct mapping of relational data to RDF. W3C RDB2RDF Working Group. Available from: <http://www.w3.org/TR/rdb-direct-mapping/> [last accessed December 13, 2010].

18. Das S, Sundara S, Cyganiak R. R2RML: RDB to RDF Mapping Language. W3C Working Draft 20 September 2011. Available from: <http://www.w3.org/TR/2011/WD-r2rml-20110920/> [last accessed August 11, 2011].
19. Das S, Sundara S, Cyganiak R. R2RML: RDB to RDF Mapping Language. W3C Working Draft 24 March 2011. Available from: <http://www.w3.org/TR/2011/WD-r2rml-20110324/> [last accessed August 11, 2011].
20. Das S, Sundara S, Cyganiak R. R2RML: RDB to RDF Mapping Language. W3C Working Draft 28 October 2010. Available from: <http://www.w3.org/TR/2010/WD-r2rml-20101028/> [last accessed August 11, 2011].
21. Carroll JJ, Dickinson I, Dollin C, Reynolds D, Seaborne A, Wilkinson K. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th World Wide Web Conference*, New York City; May 2004.
22. Picinnini H, Lemos M, Casanova MA, Furtado AL. W-Ray: a strategy to publish deep Web geographic data. In *Proceedings of the 4th International Workshop on Semantic and Conceptual Issues in GIS (SeCoGIS 2010)*. Vancouver, Canada, Springer: Berlin Heidelberg, 2010.
23. Parnas DL. On the criteria to be used in decomposing systems into modules. *Communications of the ACM* 1972; **15**(12):1053–1058.