International Journal of Database Theory and Application
Vol.7, No.6 (2014), pp.29-44
http://dx.doi.org/10.14257/ijdta.2014.7.6.04

A RIF Based Mapping of RDB2RDF

Ying Chen[1,2], Zhuoming Xu[1 (✉)], Yuyan Ni[1], Guangxu Cao[1] and Shiqing Zhang[2]
[1] *College of Computer and Information, Hohai University, Nanjing 210098, Jiangsu, P.R. China*
[2] *Institute of Image Processing and Pattern Recognition, Taizhou University, Taizhou 317000, Zhejiang, P.R. China*
ychen222@tzc.edu.cn, {zmxu, yyni_83}@hhu.edu.cn, {heromac, tzczsq}@163.com

Abstract

Mapping RDB to RDF (i.e., RDB2RDF) is the key to constructing the Semantic Web, hence has been an active research field during the last decade. Many technically heterogeneous RDB2RDF tools resulted in non-interchangeable and unreusable RDB2RDF mapping descriptions. In 2009, the W3C RDB2RDF Incubator Group Report once strongly suggested that the RDB2RDF mapping language be expressed in rules as defined by the W3C Rule Interchange Format (RIF) Working Group, because rules are an effective way to express mappings between information models, and RIF, as part of the infrastructure for the Semantic Web, is now a standard for exchanging rules among Web rule systems. This paper addresses the issue of RIF-based RDB2RDF mapping and proposes a database semantics-driven, RIF Production Rule Dialect (RIF-PRD) based mapping description approach. The work includes defining a set of generic RIF-PRD mapping rules for RDB2RDF, developing a prototype mapping engine called RIFD2RME (stands for RIF-based RDB2RDF Mapping Engine), and conducting case study experiments with the prototype. The experimental results indicate that the proposed mapping approach is achievable and effective.

Keywords: *RDB2RDF mapping; Rule Interchange Format; RIF-PRD; relational database; Resource Description Framework; Semantic Web*

1. Introduction

The Semantic Web [1] aims at constructing a Web of Data based on the Resource Description Framework (RDF) [2] data model, allowing data to be shared and reused across communities and applications. Relational Databases (RDBs) are the primary sources of data in the Web [3]. Thus mapping RDB to RDF (commonly abbreviated as *RDB2RDF*) is the key to construction of the Web of Data. Quite a few RDB2RDF techniques and tools [4-6] have been proposed over the last years. However, many early tools adopt various methods and different and proprietary mapping mechanisms and languages, making it difficult to share and reuse the mapping descriptions across applications or mapping engines.

Motivated by the interchangeableness/reusability requirements and the wider scope of use cases and requirements [7] for RDB2RDF mapping, in 2012 the W3C RDB2RDF Working Group standardized the RDB2RDF mapping mechanism and language, including *Direct Mapping* [8]—simple and automatic mapping and *R2RML* [9]—a generic language to describe customized mappings. But the Working Group did not recommend any implementation for Direct Mapping and R2RML. Lately, several existing tools or ongoing projects (cf. the W3C's RDB2RDF Implementation Report [10] and a recent survey [6]) are implementing Direct Mapping or R2RML processors, including our processor prototype [11].

Although the W3C mapping standard indicates a good prospect, suspicions (even criticisms) persist [6]. Concerning Direct Mapping, despite being automatic, direct mapping rules intend to be encoded in an application program, rather than to be interchanged across applications or mapping engines. As for R2RML, despite being interchangeable, the language may not apply to all needs in the wide scope of RDB to RDF translation applications [6]. Even worse, R2RML-based customized mappings have to be created *manually* by domain experts. Recently, a semi-automatic tool for generating customized R2RML mappings was proposed in [12], but the user still needs to draw class and property correspondence assertions (CAs) from the input (i.e., the source RDB schema and target ontology/RDF schema).

On the other hand, the W3C Rule Interchange Format (RIF) Working Group created a standard for exchanging rules among Web rule systems [13], and a family of languages, called *RIF dialects*, were published in 2013 by the Working Group. As early as 2009, the W3C RDB2RDF Incubator Group Report [14] stated "*There is a strong suggestion that the [RDB2RDF] mapping language be expressed in rules as defined by the W3C RIF WG.*" The reason for this suggestion is that rules are an effective way to express mappings between information models, and RIF, as part of the infrastructure for the Semantic Web, is a standard for exchanging rules. However, according to recent survey reports [5,6] and to our knowledge, currently there is no RDB2RDF tool available to use RIF as the mapping language.

This paper addresses the issue of RIF-based RDB2RDF mapping. We choose RIF-PRD, the RIF Production Rule Dialect [15], as the rule language to define a set of generic RDB2RDF mapping rules. As a proof-of-concept research, we focus on direct, automatic mapping that relies on a database semantics-driven approach. The mapping rules are established based upon the conceptual correspondence between the RDB schema and the RDF data model. The extracted RDB schema knowledge is stored, in the form of RIF predicates, into a priori knowledge base. By applying the defined RIF-PRD rules, reasoning against the schema knowledge can achieve a set of executable mapping rules that are stored, also in the form of RIF predicates, in a posteriori knowledge base, according to which the relational data can be mapped into an equivalent RDF dataset. Based on the above approach, we have implemented a prototype mapping engine called RIFD2RME (stands for *RIF-based RDB2RDF Mapping Engine*), and conducted case study experiments with the prototype. The experimental results indicate that the proposed mapping approach is achievable and effective.

The rest of the paper is organized as follows. Section 2 briefly discusses related work. We present our generic RIF-PRD mapping rules in Section 3. Section 4 describes the architecture of RIFD2RME, and the core design principles and implementation technologies of RIFD2RME's main components. In Section 5, we present a running example to illustrate the implementability and effectiveness of our proposed method. Finally, we concludes the work in Section 6.

2. Related Work

Making relational data accessible to the Semantic Web has been an active field of research during the last decade. As a result, many RDB2RDF techniques and tools (cf. [4-6]) have been proposed over the last years. In September 2012, the publication of the

W3C's RDB2RDF standards [8,9] has marked a new step towards the realization of the Web of Data. At the same time, facing the large variety of the RDB2RDF tools, several studies have been carried out to compare and review the different approaches and techniques, resulting in several in-depth survey reports [4-6,16,17]. Instead of providing the reader with a detailed discussion of the approaches, techniques and tools, we refer the reader to these survey reports.

In January 2009, members of the W3C RDB2RDF Incubator Group conducted a wide scope review [4], with the goal of providing the RDB2RDF WG with a comprehensive overview of the different approaches, in order to serve as a basis for the creation of R2RML. In 2011, Hert, *et al.* [16] proposed a feature-based comparison framework derived from the W3C use cases and requirements [7] and provided a comparison of the existing RDB2RDF mapping languages at that time. In 2011, Sequeda, *et al.* [17] provided a survey of directly mapping SQL databases to the Semantic Web, intending to serve as a basis for the definition of the W3C's Direct Mapping. In early 2012, Spanos, *et al.* [5] made a comprehensive review of the methods, proposed by 2011, for bringing RDBs into the Semantic Web, putting the stress on the creation and alignment of ontologies. Lately, Michel, *et al.* [6] published an online survey report on RDB2RDF translation approaches and tools, including non-R2RML tools such as D2RQ [18,19], RDBToOnto [20] and Triplify [21], as well as R2RML compliant tools such as Virtuoso RDF Views [22], Ultrawrap [23], RDF-RDB2RDF (https://metacpan.org/release/RDF-RDB2RDF), XSPARQL (http://xsparql.deri.org/), DB2Triples (https://github.com/antidot/db2triples) and Morph-RDB (https://github.com/jpcik/morph).

According to the aforementioned survey reports and our own studies, we briefly give the following findings based on the classification framework and analysis results provided by [6].

Mapping description (*direct mapping* vs. *augmented direct mapping* vs. *domain semantics-driven mapping*): Most of the tools (D2RQ, RDF-RDB2RDF, Ultrawrap and DB2Triples) apply both the direct mapping approach and the domain semantics-driven mapping approach. Triplify, Virtuoso RDF Views, XSPARQL and Morph-RDB adopt the domain semantics-driven mapping approach, while RDBToOnto applies the augmented direct mapping approach.

Mapping implementation (*data materialization* vs. *on-demand mapping*): D2RQ, Triplify, and Morph-RDB support both data materialization and on-demand mapping. RDBToOnto, RDF-RDB2RDF, XSPARQL and DB2Triples only support data materialization, while Virtuoso RDF Views and Ultrawrap only support on-demand mapping.

Data retrieval (*query-based access* vs. *linked data*): D2RQ, Virtuoso RDF Views and Ultrawrap support both query-based access and the linked data approach. XSPARQL and Morph-RDB only support query-based access, while Triplify only support the linked data approach.

The RIF Working Group was chartered by the W3C in 2005 to create a standard for exchanging rules among Web rule systems [13]. RIF focused on exchange rather than trying to develop a single one-fits-all rule language, hence a family of languages, called *RIF dialects*, with rigorously specified syntax and semantics, were published by the RIF Working Group in 2013. The RIF production rule dialect (RIF-PRD) [15] is one of a set of rule interchange dialects that also includes the RIF Core dialect (RIF-Core) and the RIF basic logic dialect (RIF-BLD). From a theoretical perspective, RIF-Core corresponds to the language of definite Horn rules without function symbols (often called “Datalog”) with a standard first-order semantics. RIF-BLD corresponds to the language of definite Horn rules with equality and a standard first-order semantics. RIF-PRD is thus a rules-with-actions dialect that supports externally defined terms, functions, predicates and atomic formulas, therefore it has sufficient expressive power to describe RDB2RDF mapping rules.

3. RIF-PRD based RDB2RDF Mapping Rules

This section introduces our RIF-PRD based RDB2RDF mapping rules. We first explain some basic concepts regarding relational database schema and the RIF-PRD rule language, then present the generic RIF-PRD mapping rules at both schema and instance levels.

3.1. Preliminaries

In this paper we restrict our attention to those aspects that constitute the core of a relational database schema. The considered relational elements include relations (a.k.a. tables), attributes (a.k.a. columns), datatypes, primary keys (PKs) and foreign keys (FKs) of relational schemas. We assume without loss of generality that all relational schemas are in the third normal form. We give a formal definition of a relational database schema in **Definition 1**.

Definition 1. A *relational database schema* is a tuple $S=(N, attr, DT, pk, fk)$, where

- N is a finite *name* set partitioned into: (1) a subset ET of *entity table/relation* names; each entity table contains rows of instance data describing entities in the real world, (2) a subset RT of *relationship table/relation* names; each relationship table contains rows of instance data describing relationships between the entities, and (3) a subset DT of *datatype* names; each datatype is a predefined RDBMS datatype, specifying a value range of the relevant instance data.
- For each $t \in ET \cup RT$, there is a finite nonempty set $col(t)=[A_1:d_1,...,A_h:d_h]$ with *column/attribute* names $c_1,...,c_h$ and their corresponding datatypes $d_1,...,d_h \in DT$.
- For each $t \in ET \cup RT$, there is exactly one *primary key* $pk(t)$ whose values uniquely determine each row of the instance data in t, where either $pk(t) \in col(t)$ (in this case $pk(t)$ is a *single-column key*) or $pk(t) \subseteq col(t)$ (in this case $pk(t)$ is a *composite key* with more than one column).
- For each $t \in ET \cup RT$, there are $n\,(n \geq 0)$ *foreign keys.* If $n \geq 1$, then the n foreign key(s) are $fk_1(t), fk_2(t), \ldots, fk_n(t) \subseteq col(t)$, where each value of the column(s) in $fk_i(t), i=1,\ldots,n$ references the relevant value of the column(s) in the primary key $pk(r)$ of another entity table $r \in ET$.

Furthermore, we can classify different types of entity tables and relationship tables by analyzing PK, FK(s), and sometimes the instance data in each table. The strategies for identifying typical table types are as follows:

- *Normal entity table:* it has one PK and no FK.

- *Entity table containing a one-to-one or one-to-many binary relationship:* it has exactly one PK and one FK, and the PK is disjoint with the FK; if many values of the PK relate to one value of the FK, then the entity table contains a one-to-many binary relationship; otherwise it contains a one-to-one binary relationship.
- *Many-to-many binary relationship table:* it has exactly one PK and two FKs, and the PK is the composite of the two FKs.
- *One-to-many/one-to-one binary relationship table:* it has exactly one PK and two FKs, and the first FK is also the PK whereas the intersection of the second FK with the PK is an empty set; if many values of the PK relate to one value of the second FK, then the table is a one-to-many binary relationship table; otherwise it is a one-to-one binary relationship table.
- *n-ary relationship table:* it has one PK and more than two FKs, and the PK is the composite of these FKs.

Following the above strategies, we can even design an algorithm to achieve automatic table type identification after extraction of the relational database schema.

As for the rule language, we choose RIF-PRD [15], a rules-with-actions dialect, to specify the mapping rules from relational database to RDF. Production rules have an *if* part, or *condition*, and a *then* part, or *action*. The condition is like the condition part of logic rules. The then part contains actions that can assert facts, modify facts, retract facts, and have other side-effects. RIF-PRD has a concise abstract syntax specified in mathematical English, and a concrete XML syntax used for practical applications on the Web. The two syntaxes have a good correspondence. We will use the abstract syntax to present the mapping rules for saving space, while the mapping engine, RIFD2RME, actually uses the concrete syntax of RIF-PRD.

Additionally, the vocabularies used in the RIF-PRD based mapping rules come from multiple namespaces including RIF built-in functions and predicates, XML Schema, and the namespaces for RIF's externally specified predicates and functions (see Tables 1 and 2) introduced by ourselves. These namespace prefixes are declared as follows.

```
Prefix(func <http://www.w3.org/2007/rif-builtin-function#>)
Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>)
Prefix(xs  <http://www.w3.org/2001/XMLSchema#>)
Prefix(ex  <http://www.hhu.edu.cn/RIF-mapping/>)
Prefix(fc  <http://www.hhu.edu.cn/RIF-function/>)
```

Table 1. Predicates and Functions Used in the Schema-mapping RIF-PRD Rules

Symbols	**Descriptions**
RDB-related predicates	
`table(?t)`	`?t` is a table (a.k.a. relation).
`entity_table(?t)`	`?t` is a normal entity table that has one PK and no FK.
`relationship_table(?t ?t1 ?t2...?tn)`	`?t` is a binary or *n*-ary relationship table that contains exactly two or more FKs referencing other (ordinarily entity) tables and does not contain any non-FK columns.
`col(?c ?t ?c_type)`	`?c` is a column (a.k.a. attribute) of table `?t`, and `?c`'s datatype is `?c_type`.

`pk(?p ?t)`	`?p` is the PK of table `?t`. In the actual situation, we use RIF built-in predicate `List(c1...cm)`, `m>=1`, `c1...cm` are columns of the table, to represent a single-attribute key or composite key.
`fk(?f ?t ?p ?s)`	`?f` is a FK of table `?t`, which references the value of PK `?p` of table `?s`. Both take the form of `List(c1...cm)`.
`nonFK(?c ?t ?c_type)`	`?c` is a non-FK column of table `?t`, and `?c`'s datatype is `?c_type`.
RDF-related predicates	
`class(?t)`	Table `?t` is mapped to an RDF class.
`object_p(?op ?t ?s)`	FK or binary relationship table `?op` is mapped to such an RDF object property that its domains is the RDF class corresponding to table `?t` and its range is the RDF class corresponding to table `?s`.
`datatype_p(?c ?t ?mapped_xsd_type)`	Non-FK column `?c` of table `?t` is mapped to such an RDF datatype property that its domain is the RDF class corresponding to `?t` and its ranges is the XML Schema datatype, `?mapped_xsd_type`, corresponding to `?c`'s datatype. The datatype mapping is achieved through RIF's externally specified function `type()` given below.
Functions	
`type(?c_type)`	This function returns the mapped XML Schema datatype of a relational column datatype `?c_type`.

Table 2. Predicates and Functions Used in the Instance-mapping RIF-PRD Rules

Symbols	**Descriptions**
Instance mapping related predicate	
`triple(?s ?p ?o)`	`?s`, `?p`, and `?o` together construct an RDF triple whose subject is represented by `?s`, predicate `?p`, and object `?o`. This predicate is used to instruct the mapping engine to translate relational data into RDF data.
Instance mapping related functions	
`generateTableURI(?t)`	This function returns a URI that identifies the RDF class corresponding to table `?t`. The URI is generated by connecting a base URI and the table name.
`generateColumnURI(?t ?c)`	This function returns a URI that identifies the RDF properties corresponding to one column or one group of columns, `?c`, of table `?t`. The URI is generated by connecting a base URI, the table name, and name(s) of the column(s).
`generateRowURI(?t ?p ?v)`	This function returns a URI that identifies an instance of the RDF class corresponding to table `?t`. Each row of table `?t` is uniquely determined by the value, `?v`, of this table's PK, `?p`, therefore, all row-relevant instances will be uniquely created. The URI is generated by connecting a base URI, the table name, and the PK's column name(s) and value(s).
`data(?t ?c)`	This function returns the value(s) of column `?c` in table `?t`. The data retrieval is achieved by executing a SQL query.

3.2. Schema Mapping Rules

The following RIF-PRD rules, **Rules 1-5**, are used to mapping an RDB schema to an RDF schema (i.e., lightweight ontology), which will structurally organize the mapped instance data (i.e., RDF data).

Rule 1: If a table, ?t, has one PK and does not contain any FK, then ?t is a normal entity table. The RIF-PRD rule in the abstract syntax is as follows:

```
Forall ?t such that (
   If And(ex:table(?t)
              Exists ?p (ex:pk(?p ?t))
              Not Exists ?f ?p ?s (ex:fk(?f ?t ?p ?s)))
   Then Assert(ex:entity_table(?t)) )
```

Rule 2: If a table, ?t, contains exactly two or more FKs that reference other tables (ordinarily entity tables) and does not contain any non-FK columns, then ?t is a binary or *n*-ary relationship table. The RIF-PRD rule in the abstract syntax is as follows:

```
Forall ?t ?t1 ... ?tn such that (
   If And(ex:table(?t)
              Exists ?f1 ?p1 ... ?fn ?pn (
                 And(ex:fk(?f1 ?t ?p1 ?tl) ... ex:fk(?fn ?t ?pn ?tn)
                       External(pred:literal-not-identical(?f1 ... ?fn))))
              Not Exists ?c (ex:nonFK(?c ?t ?c_type)))
   Then Assert(ex:relationship_table(?t ?t1 ... ?tn)) )
```

Rule 3: If a table, ?t, is an entity table or is a relationship table that also contains at least one non-FK column, then ?r is mapped to an RDF class. The RIF-PRD rule in the abstract syntax is as follows:

```
Forall ?t such that (
   If And(ex:entity_table(?t)
              Not Exists ?t1 ... ?tn (ex:relationship_table(?t ?t1 ... ?tn)))
   Then Assert(ex:class(?t)) )
```

Rule 4: If a table, ?t, is a binary relationship table (without any non-FK columns), then ?t can be mapped to such an RDF object property (rather than an RDF class) that its domain is the RDF class corresponding to one ?t-referenced entity table and its ranges is the RDF class corresponding to another ?t-referenced entity table. The RIF-PRD rule in the abstract syntax is as follows:

```
Forall ?t ?t1 ?t2 such that (
   If And(ex:relationship_table(?t ?t1 ?t2)
               ex:entity_table(?t1)
               ex:entity_table(?t2))
   Then Assert(ex:object_p(?t ?t1 ?t2)) )
```

Rule 5: If a table, ?t, is an entity table or a relationship table that contains non-FK columns, and all ?t's FKs reference other entity tables' PKs, then all these FKs are mapped to such RDF object properties that their domains are the RDF class corresponding to ?t and their ranges are the RDF classes corresponding to the referenced entity tables. The RIF-PRD rule in the abstract syntax is as follows:

```
Forall ?t ?t1 ... ?tn ?f1 ... ?fn such that (
   If And(ex:table(?t ?t1 ... ?tn)
              ex:entity_table(?t1) ... ex:entity_table(?tn)
              Exists ?f1 ?p1 ... ?fn ?pn (
                 And(ex:fk(?f1 ?t ?p1 ?t1) ... ex:fk(?fn ?t ?pn ?tn)))
   Then Assert(ex:object_p(?f1 ?t ?t1))... Assert(ex:object_p(?fn ?t ?tn)))
```

Rule 6: All the non-FK columns in any type of table, ?t, are mapped to such RDF datatype properties that their domains are the RDF class corresponding to ?t and their ranges are the XML Schema datatypes corresponding to the datatype of the non-FK columns. The RIF-PRD rule in the abstract syntax is as follows:

```
Forall ?c ?t such that (
    If ex:nonFK(?c ?t ?c_type)
    Then Assert(ex:datatype_p(?c ?t External(fc:type(?c_type)))) )
```

3.3. Instance Mapping Rules

The following RIF-PRD rules, **Rules 7-10**, are used to mapping the relational data to RDF data according to the generated RDF schema.

Rule 7: If a table, ?t, can be mapped to an RDF class, then for each row of the table, rdf:type should be used, as a predicate, to create an RDF triple, stating that the row is an instance of the RDF class. The RIF-PRD rule in the abstract syntax is as follows:

```
Forall ?t(
    If And(ex:class(?t)
              Exists ?p(ex:pk(?p ?t)))
    Then Excute(ex:triple(fc:generateRowURI(?t ?p ex:data(?t ?p)) “rdf:type”
                                   fc:generateTableURI(?t))) )
```

Rule 8: If an column, c, of table t can be mapped to an RDF datatype property, then for each row of the table, this datatype property should be use, as a predicate, to create an RDF triple, stating that the row-related instance has a datatype property whose value equals to the column’s value with the mapped XML Schema datatype. The RIF-PRD rule in the abstract syntax is as follows:

```
Forall ?c ?t ?c_type(
    If And(datatype_p(?c ?t External(fc:type(?c_type))
              Exists ?p (ex:pk(?p ?t)))
    Then Excute(ex:triple(fc:generateRowURI(?t ?p ex:data(?t ?p))
                          fc:generateColumnURI(?t ?c)
                          External(func:concat(ex:data(?t ?c)”^^”
                                               External(fc:type(?c_type))))) )
```

Rule 9: If an column, c, of table t can be mapped to an RDF object property, then for each row of the table, this object property should be use, as a predicate, to create an RDF triple, stating that the row-related instance has an object property whose value equals to the related instance of an RDF class corresponding to the t-referenced table. The RIF-PRD rule in the abstract syntax is as follows:

```
Forall ?f ?t ?t1(
    If And(ex:object_p(?f ?t ?t1)
              Exists ?p1 (ex:pk(?p1 ?t))
              Exists ?p2 (And(ex:pk(?p2 ?t1)
                              ex:fk(?f ?t ?p2 ?t1))))
    Then Excute(ex:triple(fc:generateRowURI(?t ?p1 ex:data(?t ?p1))
                              fc:generateColumnURI(?t ?f)
                              fc:generateRowURI(?t1 ?p2 ex:data(?t1 ?p2)))) )
```

Rule 10: If a binary relationship table, ?t, can be mapped to an RDF object property, and ?t’s two related entity tables are ?t1 and ?t2, then this object property should be use, as a predicate, to create RDF triples that connect the instances of ?t1’s class to the instances of ?t2’s class according to the data in table ?t. The RIF-PRD rule in the abstract syntax is as follows:

```
Forall ?t ?t1 ?t2(
    If And(ex:relationship_table(?t ?t1 ?t2)
              ex:object_p(?t ?t1 ?t2)
              Exists ?p1(ex:pk(?p1 ?t1))
              Exists ?p2(ex:pk(?p2 ?t2)))
    Then Excute(ex:triple(fc:generateRowURI(?t1 ?p1 ex:data(?t1 ?p1))
                              fc:generateTableURI(?t)
                              fc:generateRowURI(?t2 ?p2 ex:data(?t2 ?p2)))) )
```

4. Prototype Implementation of RIFD2RME

To verify the effectiveness of the RIF-PRD mapping rules and the implementability of our proposed mapping method, we have designed and developed a prototype mapping engine called RIFD2RME (stands for RIF-based RDB2RDF Mapping Engine). This section briefly describes the architecture and main components of RIFD2RME.

4.1. Architecture

Figure 1 illustrates the architecture and the RIF-based RDB2RDF mapping process of RIFD2RME. The mapping engine takes a relational database (RDB) and an RIF-PRD document (specifying RDB2RDB schema and instance mapping rules) as input, and produces the mapped RDF dataset as output. The resulting RDF data can then be consumed *either* by SPARQL [24] query clients *or* by HTTP-based browser/crawler clients. RIFD2RME's main components are briefly described as follows.

- *RDB Schema Extractor* performs extraction of the schema elements as defined in **Definition 1** from the RDB, analysis of the schema information, and finally storing of the obtained knowledge, in the form of RIF predicates, into *A Priori Knowledge Base*.
- *RIF-PRD Rules Parser* is used to parse an RIF-PRD document in the XML concrete syntax, encapsulate the RIF-PRD rule elements in Java objects and pass the Java objects to the next component.
- *RIF-PRD Reasoner* performs reasoning against the schema knowledge based on the defined RIF-PRD rules to achieve a set of executable mapping rules that are stored, in the form of RIF predicates, in *A Posteriori Knowledge Base*, according to which the relational data can be mapped into RDF data.
- *RDB2RDF Mapper* performs translation from the RDB to an equivalent *RDF dataset* according to the executable mapping rules.

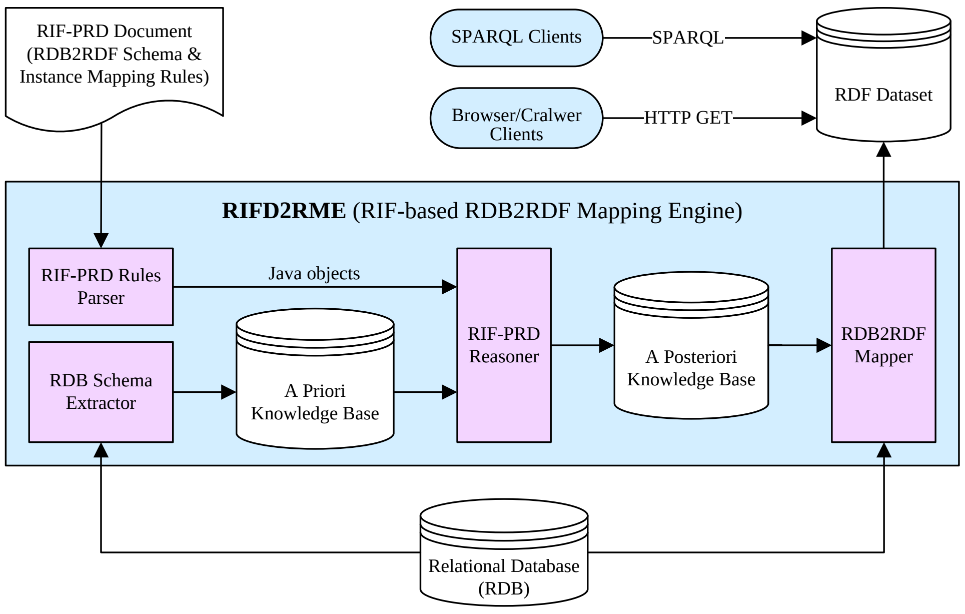


Figure 1. RIFD2RME's Architecture and Mapping Process

The core design principles and implementation technologies of these components will be explained in brief in the subsequent subsections.

4.2. RDB Schema Extractor

The *RDB Schema Extractor* employs **Algorithm 1** to extract schema elements, including tables, columns, datatypes, primary keys and foreign keys, from the relational database and put the obtained knowledge, in the form of RIF predicates, into *A Priori Knowledge Base*.

Algorithm 1 has been implemented with Java language (J2SDK 1.6.0), where database connection (Step 1) was implemented with java.sql.DriverManager.getConnection and extraction of schema elements (Step 2) was achieved through java.sql.DatabaseMetaData.

```
Algorithm 1. RDBSchemaExtraction(RDB)
Input: relational database RDB.
Output: RIF predicate set P that is stored in A Priori Knowledge Base.
Steps:
1:   connect to relational database RDB;
2:   get metadata from RDB;
3:   put all tables from metadata into tableSet;
4:   initialize $P \leftarrow \varnothing$;
5:   for each table $t \in tableSet$ do
6:       generate a predicate, table(t), and put it into P;
7:       for each column c of t do
8:           generate a predicate, col(c t c_type), and put it into P;
9:           if (c does not belong to any foreign key of t) then
10:              generate a predicate, nonFK(c t c_type), and put it into P;
11:      end for;
12:      if (t contains a primary key consisting of t's column(s) c1...cm, m>=1) then
13:          generate a predicate, pk(List(c1...cm) t), and put it into P;
14:      if t contains foreign keys then
15         for each foreign key, consisting of t's column(s) c1...cn, n>=1, which references
                the value of primary key p of table s do
16:              generate a predicate fk(List(c1...cn) t p s), and put it into P;
17:  end for.
```

4.3. RIF-PRD Rules Parser

As mentioned earlier, all the RIF-PRD based mapping rules are actually stored in an RIF-PRD document in the XML concrete syntax specified in the W3C document [15]. *RIF-PRD Rules Parser*'s overall process flow of parsing an RIF-PRD document is shown in Figure 2.

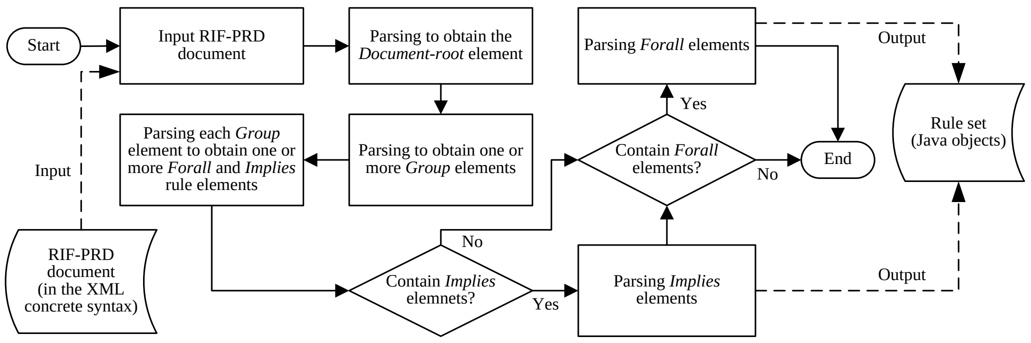

Start
Input RIF-PRD document
Parsing to obtain the Document-root element
Parsing Forall elements
Output
Yes
Rule set (Java objects)
Parsing each Group element to obtain one or more Forall and Implies rule elements
Parsing to obtain one or more Group elements
Contain Forall elements?
End
No
Input
No
RIF-PRD document (in the XML concrete syntax)
Contain Implies elemnets?
Yes
Parsing Implies elements
Output

Figure 2. Overall Process Flow of Parsing an RIF-PRD Document

The parser has been implemented as a Java program based on Java XML API dom4j. Each RIF element such as *Group*, *Forall*, and *Implies* in the RIF-PRD rules was packaged as a Java object and passed to *RIF-PRD Reasoner*.

4.4. RIF-PRD Reasoner

Based on the RIF-PRD rules, reasoning against the schema knowledge in *A Priori Knowledge Base* will achieve a set of executable mapping rules stored in *A Posteriori Knowledge Base*. All the RIF-PRD rules are defined using the *Forall* construct, representing rules with bound variables. The overall process flow of reasoning an RIF-PRD *Forall* rule is depicted in Figure 3. *RIF-PRD Reasoner* has been implemented as a Java program.

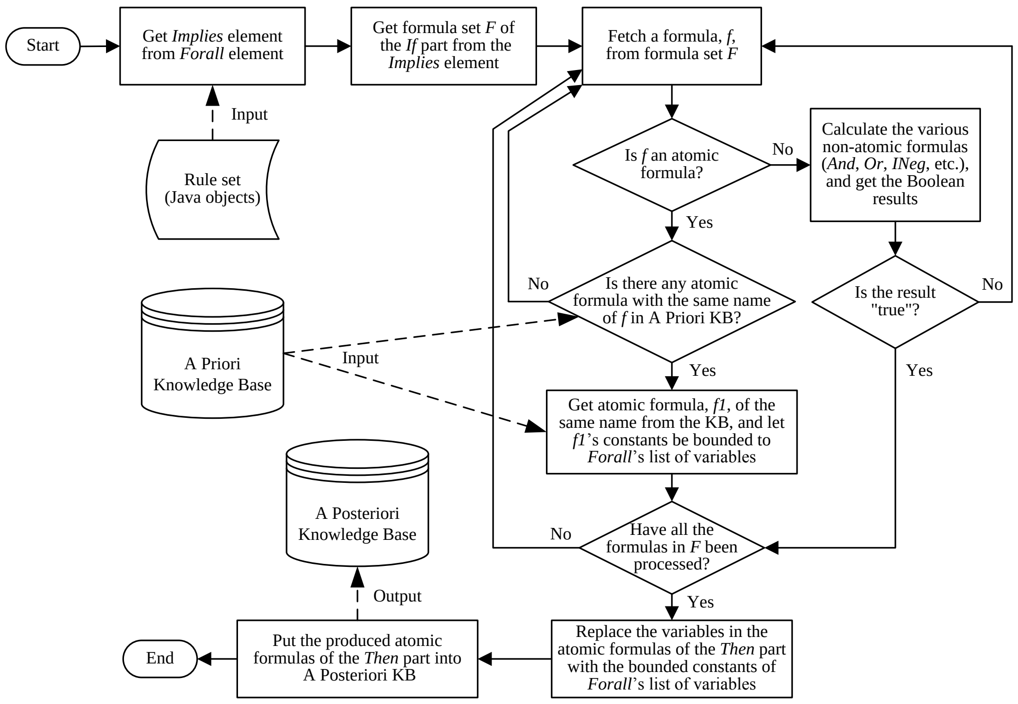

Start
Get Implies element from Forall element
Get formula set F of the If part from the Implies element
Fetch a formula, f, from formula set F
Input
Rule set (Java objects)
Is f an atomic formula?
No
Calculate the various non-atomic formulas (And, Or, INeg, etc.), and get the Boolean results
Yes
No
Is there any atomic formula with the same name of f in A Priori KB?
Is the result "true"?
No
A Priori Knowledge Base
Input
Yes
Yes
Get atomic formula, f1, of the same name from the KB, and let f1's constants be bounded to Forall's list of variables
A Posteriori Knowledge Base
No
Have all the formulas in F been processed?
Output
Yes
End
Put the produced atomic formulas of the Then part into A Posteriori KB
Replace the variables in the atomic formulas of the Then part with the bounded constants of Forall's list of variables

Figure 3. Overall Process Flow of Reasoning an RIF-PRD Forall Rule

4.5. RDB2RDF Mapper

Although our RIF-PRD rules can support *both* the data materialization approach (i.e., ETL-based RDF dumping) *and* the on-demand mapping approach (i.e., query-driven, dynamic retrieval), at the present stage we only implemented the data materialization process in the prototype mapping engine.

Following the executable mapping rules, *RDB2RDF Mapper*, implemented as a Java program, performs translation from the relational data to an equivalent RDF dataset. All the triple predicates in the instance mapping rules will be executed during the data translation process. The overall process flow of a triple predicate is as follows:

Step 1: Get a triple predicate from *A Posteriori Knowledge Base*;

Step 2: Create and execute SQL SELECT statements corresponding to triple predicate's three parameters to generate three URIs identifying the subject, predicate, and object resources of an RDF triple.

Step 3: Store the produced RDF triple in the RDF dataset (e.g., triplestore).

According to the defined RIF-PRD functions and the specific bounded tables and columns, the SQL SELECT statement is *either* a simple select conditional query against a single table on the specific column(s) *or* a join conditional query over multiple tables.

5. Running Example

We have used the prototype mapping engine, RIFD2RME, to conduct several case studies. All case studies have obtained a correct, satisfactory outcome. Saving space, in the following we illustrate a small running example (see Figures 4, 5, 6, and 7).

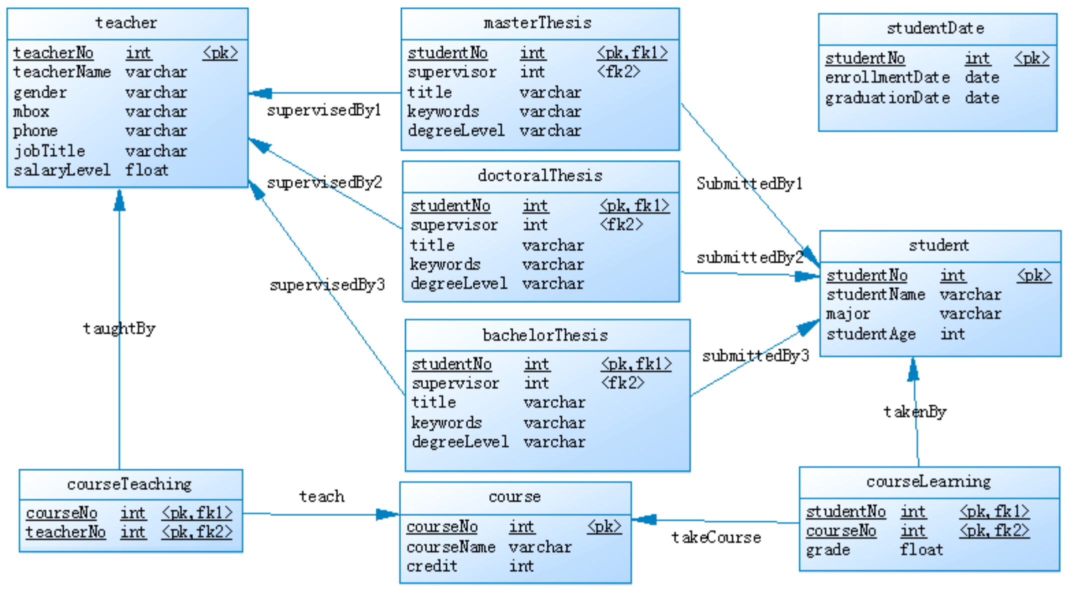
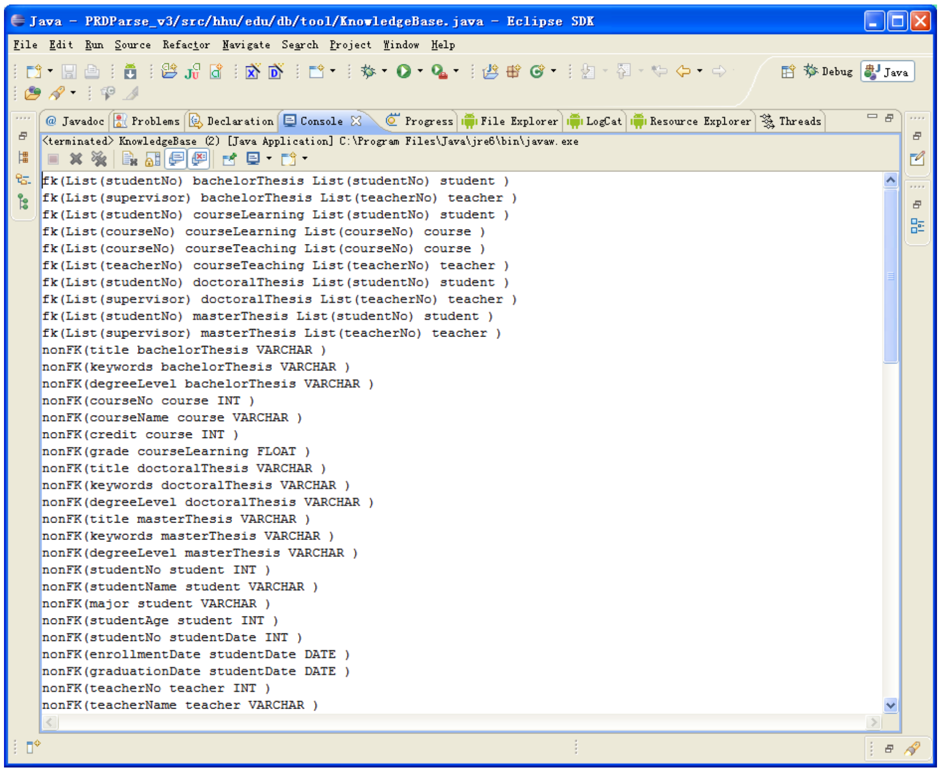
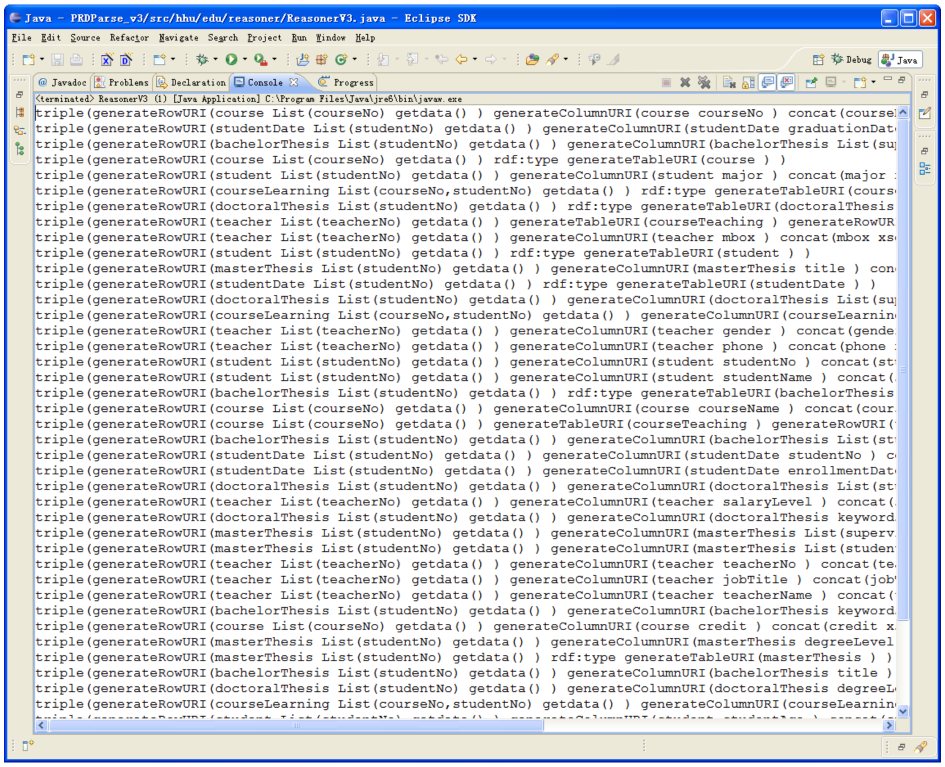


Figure 4. Physical Data Model (PDM) of the University Relational Database



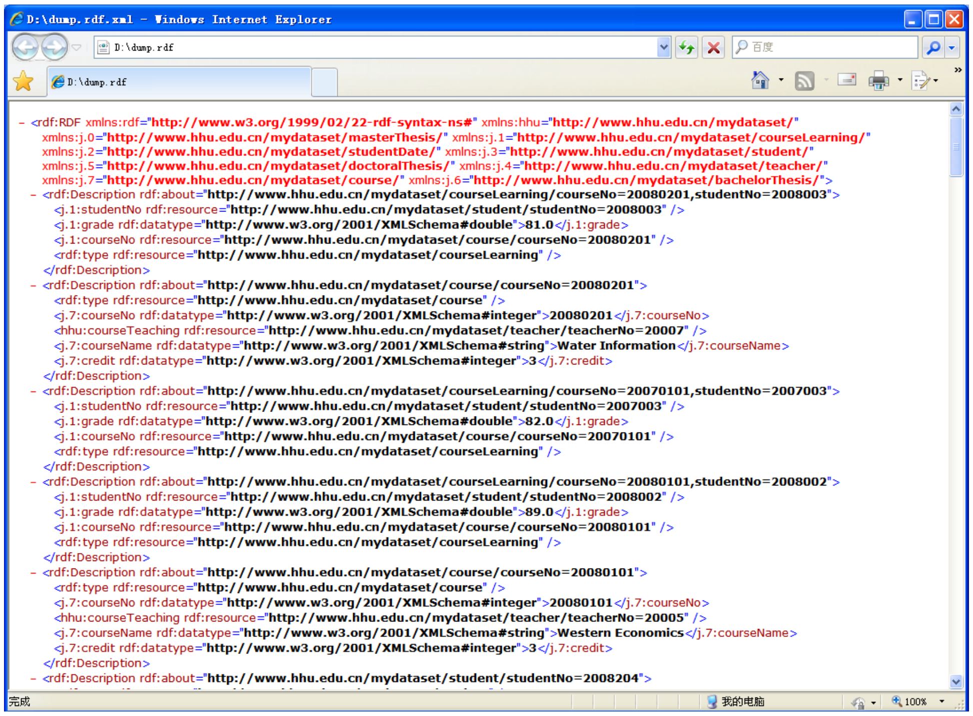
```
<terminated> KnowledgeBase (2) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe
fk(List(studentNo) bachelorThesis List(studentNo) student )
fk(List(supervisor) bachelorThesis List(teacherNo) teacher )
fk(List(studentNo) courseLearning List(studentNo) student )
fk(List(courseNo) courseLearning List(courseNo) course )
fk(List(courseNo) courseTeaching List(courseNo) course )
fk(List(teacherNo) courseTeaching List(teacherNo) teacher )
fk(List(studentNo) doctoralThesis List(studentNo) student )
fk(List(supervisor) doctoralThesis List(teacherNo) teacher )
fk(List(studentNo) masterThesis List(studentNo) student )
fk(List(supervisor) masterThesis List(teacherNo) teacher )
nonFK(title bachelorThesis VARCHAR )
nonFK(keywords bachelorThesis VARCHAR )
nonFK(degreeLevel bachelorThesis VARCHAR )
nonFK(courseNo course INT )
nonFK(courseName course VARCHAR )
nonFK(credit course INT )
nonFK(grade courseLearning FLOAT )
nonFK(title doctoralThesis VARCHAR )
nonFK(keywords doctoralThesis VARCHAR )
nonFK(degreeLevel doctoralThesis VARCHAR )
nonFK(title masterThesis VARCHAR )
nonFK(keywords masterThesis VARCHAR )
nonFK(degreeLevel masterThesis VARCHAR )
nonFK(studentNo student INT )
nonFK(studentName student VARCHAR )
nonFK(major student VARCHAR )
nonFK(studentAge student INT )
nonFK(studentNo studentDate INT )
nonFK(enrollmentDate studentDate DATE )
nonFK(graduationDate studentDate DATE )
nonFK(teacherNo teacher INT )
nonFK(teacherName teacher VARCHAR )
```

Figure 5. Screenshot of the Result Produced by RDB Schema Extractor



```
<terminated> ReasonerV3 (1) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe
triple(generateRowURI(course List(courseNo) getdata() ) generateColumnURI(course courseNo ) concat(course
triple(generateRowURI(studentDate List(studentNo) getdata() ) generateColumnURI(studentDate graduationDat
triple(generateRowURI(bachelorThesis List(studentNo) getdata() ) generateColumnURI(bachelorThesis List(su
triple(generateRowURI(course List(courseNo) getdata() ) rdf:type generateTableURI(course ) )
triple(generateRowURI(student List(studentNo) getdata() ) generateColumnURI(student major ) concat(major
triple(generateRowURI(courseLearning List(courseNo,studentNo) getdata() ) rdf:type generateTableURI(cours
triple(generateRowURI(doctoralThesis List(studentNo) getdata() ) rdf:type generateTableURI(doctoralThesis
triple(generateRowURI(teacher List(teacherNo) getdata() ) generateTableURI(courseTeaching ) generateRowUR
triple(generateRowURI(teacher List(teacherNo) getdata() ) generateColumnURI(teacher mbox ) concat(mbox xs
triple(generateRowURI(student List(studentNo) getdata() ) rdf:type generateTableURI(student ) )
triple(generateRowURI(masterThesis List(studentNo) getdata() ) generateColumnURI(masterThesis title ) con
triple(generateRowURI(studentDate List(studentNo) getdata() ) rdf:type generateTableURI(studentDate ) )
triple(generateRowURI(doctoralThesis List(studentNo) getdata() ) generateColumnURI(doctoralThesis List(su
triple(generateRowURI(courseLearning List(courseNo,studentNo) getdata() ) generateColumnURI(courseLearnin
triple(generateRowURI(teacher List(teacherNo) getdata() ) generateColumnURI(teacher gender ) concat(gende
triple(generateRowURI(teacher List(teacherNo) getdata() ) generateColumnURI(teacher phone ) concat(phone
triple(generateRowURI(student List(studentNo) getdata() ) generateColumnURI(student studentNo ) concat(st
triple(generateRowURI(student List(studentNo) getdata() ) generateColumnURI(student studentName ) concat(
triple(generateRowURI(bachelorThesis List(studentNo) getdata() ) rdf:type generateTableURI(bachelorThesis
triple(generateRowURI(course List(courseNo) getdata() ) generateColumnURI(course courseName ) concat(cour
triple(generateRowURI(course List(courseNo) getdata() ) generateTableURI(courseTeaching ) generateRowURI(
triple(generateRowURI(bachelorThesis List(studentNo) getdata() ) generateColumnURI(bachelorThesis List(st
triple(generateRowURI(studentDate List(studentNo) getdata() ) generateColumnURI(studentDate studentNo ) c
triple(generateRowURI(studentDate List(studentNo) getdata() ) generateColumnURI(studentDate enrollmentDat
triple(generateRowURI(doctoralThesis List(studentNo) getdata() ) generateColumnURI(doctoralThesis List(st
triple(generateRowURI(teacher List(teacherNo) getdata() ) generateColumnURI(teacher salaryLevel ) concat(
triple(generateRowURI(doctoralThesis List(studentNo) getdata() ) generateColumnURI(doctoralThesis keyword
triple(generateRowURI(masterThesis List(studentNo) getdata() ) generateColumnURI(masterThesis List(superv
triple(generateRowURI(masterThesis List(studentNo) getdata() ) generateColumnURI(masterThesis List(studen
triple(generateRowURI(teacher List(teacherNo) getdata() ) generateColumnURI(teacher teacherNo ) concat(te
triple(generateRowURI(teacher List(teacherNo) getdata() ) generateColumnURI(teacher jobTitle ) concat(job
triple(generateRowURI(teacher List(teacherNo) getdata() ) generateColumnURI(teacher teacherName ) concat(
triple(generateRowURI(bachelorThesis List(studentNo) getdata() ) generateColumnURI(bachelorThesis keyword
triple(generateRowURI(course List(courseNo) getdata() ) generateColumnURI(course credit ) concat(credit x
triple(generateRowURI(masterThesis List(studentNo) getdata() ) generateColumnURI(masterThesis degreeLevel
triple(generateRowURI(masterThesis List(studentNo) getdata() ) rdf:type generateTableURI(masterThesis ) )
triple(generateRowURI(bachelorThesis List(studentNo) getdata() ) generateColumnURI(bachelorThesis title )
triple(generateRowURI(doctoralThesis List(studentNo) getdata() ) generateColumnURI(doctoralThesis degreeL
triple(generateRowURI(courseLearning List(courseNo,studentNo) getdata() ) generateColumnURI(courseLearnin
```

Figure 6. Screenshot of the Result Produced by RIF-PRD Reasoner



```
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:hhu="http://www.hhu.edu.cn/mydataset/"
   xmlns:j.0="http://www.hhu.edu.cn/mydataset/masterThesis/" xmlns:j.1="http://www.hhu.edu.cn/mydataset/courseLearning/"
   xmlns:j.2="http://www.hhu.edu.cn/mydataset/studentDate/" xmlns:j.3="http://www.hhu.edu.cn/mydataset/student/"
   xmlns:j.5="http://www.hhu.edu.cn/mydataset/doctoralThesis/" xmlns:j.4="http://www.hhu.edu.cn/mydataset/teacher/"
   xmlns:j.7="http://www.hhu.edu.cn/mydataset/course/" xmlns:j.6="http://www.hhu.edu.cn/mydataset/bachelorThesis/">
 - <rdf:Description rdf:about="http://www.hhu.edu.cn/mydataset/courseLearning/courseNo=20080201,studentNo=2008003">
     <j.1:studentNo rdf:resource="http://www.hhu.edu.cn/mydataset/student/studentNo=2008003" />
     <j.1:grade rdf:datatype="http://www.w3.org/2001/XMLSchema#double">81.0</j.1:grade>
     <j.1:courseNo rdf:resource="http://www.hhu.edu.cn/mydataset/course/courseNo=20080201" />
     <rdf:type rdf:resource="http://www.hhu.edu.cn/mydataset/courseLearning" />
   </rdf:Description>
 - <rdf:Description rdf:about="http://www.hhu.edu.cn/mydataset/course/courseNo=20080201">
     <rdf:type rdf:resource="http://www.hhu.edu.cn/mydataset/course" />
     <j.7:courseNo rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">20080201</j.7:courseNo>
     <hhu:courseTeaching rdf:resource="http://www.hhu.edu.cn/mydataset/teacher/teacherNo=20007" />
     <j.7:courseName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Water Information</j.7:courseName>
     <j.7:credit rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">3</j.7:credit>
   </rdf:Description>
 - <rdf:Description rdf:about="http://www.hhu.edu.cn/mydataset/courseLearning/courseNo=20070101,studentNo=2007003">
     <j.1:studentNo rdf:resource="http://www.hhu.edu.cn/mydataset/student/studentNo=2007003" />
     <j.1:grade rdf:datatype="http://www.w3.org/2001/XMLSchema#double">82.0</j.1:grade>
     <j.1:courseNo rdf:resource="http://www.hhu.edu.cn/mydataset/course/courseNo=20070101" />
     <rdf:type rdf:resource="http://www.hhu.edu.cn/mydataset/courseLearning" />
   </rdf:Description>
 - <rdf:Description rdf:about="http://www.hhu.edu.cn/mydataset/courseLearning/courseNo=20080101,studentNo=2008002">
     <j.1:studentNo rdf:resource="http://www.hhu.edu.cn/mydataset/student/studentNo=2008002" />
     <j.1:grade rdf:datatype="http://www.w3.org/2001/XMLSchema#double">89.0</j.1:grade>
     <j.1:courseNo rdf:resource="http://www.hhu.edu.cn/mydataset/course/courseNo=20080101" />
     <rdf:type rdf:resource="http://www.hhu.edu.cn/mydataset/courseLearning" />
   </rdf:Description>
 - <rdf:Description rdf:about="http://www.hhu.edu.cn/mydataset/course/courseNo=20080101">
     <rdf:type rdf:resource="http://www.hhu.edu.cn/mydataset/course" />
     <j.7:courseNo rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">20080101</j.7:courseNo>
     <hhu:courseTeaching rdf:resource="http://www.hhu.edu.cn/mydataset/teacher/teacherNo=20005" />
     <j.7:courseName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Western Economics</j.7:courseName>
     <j.7:credit rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">3</j.7:credit>
   </rdf:Description>
 - <rdf:Description rdf:about="http://www.hhu.edu.cn/mydataset/student/studentNo=2008204">
```

Figure 7. Screenshot of the Result Produced by RDB2RDF Mapper

A University RDB was created with MySQL 5.1. Figure 4 shows the physical data model (PDM), produced by PowerDesigner 12.5, of this database. Instance data have been inserted into the database tables.

RIFD2RME's *RDB Schema Extractor* performed extraction of the schema elements and generated the schema knowledge in the form of RIF predicates. A screenshot of the result produced by RDB Schema Extractor is shown in Figure 5.

RIF-PRD Reasoner performed reasoning against the schema knowledge based on the parsed RIF-PRD mapping rules and achieved a set of executable mapping rules, as shown in Figure 6.

RDB2RDF Mapper performed translation from the University relational database to an RDF dataset according to the executable mapping rules. The resulting RDF file was opened in the Windows Internet Explorer and its screenshot is shown in Figure 7.

6. Conclusion

Relational databases are the most popular data management solution for various kinds of information systems. Making relational data accessible to the Semantic Web has been recognized as a key factor in both creating the Web of Data and generating RDF data for Semantic Web applications. RDB2RDF mapping has been an active field of research during the last decade. The publication of the W3C Direct Mapping and R2RML standard in 2012 has marked a new step towards the success of the RDB2RDF effort. However, the W3C standard is still insufficient in terms of description and interchangeableness of the mapping rules, especially in situations where complex mappings and broad interchange are needed. RIF, on the other hand, has sufficient power of expression and is suitable for describing exchangeable mapping rules. In this paper, we propose a database semantics-driven, RIF-PRD based mapping description approach. Based on a set of generic RIF-PRD mapping rules, we developed a prototype mapping engine (RIFD2RME), and conducted case study experiments with the prototype. The experimental results indicate that the proposed mapping approach is achievable and effective.

Our study is only a proof-of-concept research at the present stage. The RIF-PRD mapping rules are just for augmented direct mapping from RDB to RDF, and the prototype mapping engine only supports data materialization. Future work will focus on improving the approach and extending the implementation.

Acknowledgements

This work was partially supported by the following research grants: (1) No. 61272261 and No. 61203257 from the Natural Science Foundation of China, (2) No. BK20141420 and No. BK20140857 from the Natural Science Foundation of Jiangsu Province, China.

References

[1] N. Shadbolt, T. Berners-Lee, and W. Hall, "The Semantic Web revisited," IEEE Intelligent Systems, vol. 21, no. 3, **(2006)**, pp. 96-101.
[2] F. Manola and E. Miller (Eds.), RDF Primer, W3C Recommendation, 10 February **2004**, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.
[3] B. He, M. Patel, Z. Zhang and K. Chang, "Accessing the deep Web", Communications of the ACM, vol. 50, no. 5, **(2007)**, pp. 94-101.
[4] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau Jr, S. Auer, J. Sequeda, and A. Ezzat, A Survey of Current Approaches for Mapping of Relational Databases to RDF, W3C RDB2RDF Incubator Group, 08 Jan **2009**, http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf.
[5] D.-E. Spanos, P. Stavrou, and N. Mitrou, "Bringing relational databases into the semantic web: a survey," Semantic Web Journal, vol. 3, no. 2, **(2012)**, pp. 169-209.
[6] F. Michel, J. Montagnat, and C. Faron, A survey of RDB to RDF translation approaches and tools, Tech. **(2014)**.
[7] E. Prud'hommeaux and M. Hausenblas (Eds.), "Use Cases and Requirements for Mapping Relational Databases to RDF", W3C Working Draft, **(2010)**.
[8] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda (Eds.), "A Direct Mapping of Relational Data to RDF", W3C Recommendation, **(2012)**, http://www.w3.org/TR/rdb-direct-mapping/.
[9] S. Das, S. Sundara and R. Cyganiak (Eds.), R2RML: RDB to RDF Mapping Language, W3C Recommendation, **(2012)**, http://www.w3.org/TR/r2rml.
[10] B. Villazón-Terrazas and M. Hausenblas, "RDB2RDF Implementation Report", W3C Working Group Note, **(2012)**, http://www.w3.org/TR/rdb2rdf-implementations/.
[11] S. Zhou, Z. Xu, Y. Ni, and H. Zhang, "R2RML processor for materializing RDF view of relational data: algorithms and experiments," Proceedings of the 10th Web Information System and Application Conference, Yangzhou, China, 1-3 November 2013, IEEE Computer Society **(2013)**, pp. 450-455.
[12] L. E. T. Neto, V. M. P. Vidal, M. A. Casanova, and J. M. Monteiro, "R2RML by assertion: a semi-automatic tool for generating customised R2RML mappings," The Semantic Web: ESWC 2013 Satellite Events, LNCS, vol. 7955, Springer Berlin Heidelberg, **(2013)**, pp 248-252.
[13] M. Kifer and H. Boley, "RIF Overview (Second Edition)", W3C Working Group Note, **(2013)**, http://www.w3.org/TR/rif-overview/.
[14] A. Malhotra, "W3C RDB2RDF Incubator Group Report", W3C Incubator Group Report, **(2009)**, http://www.w3.org/2005/Incubator/rdb2rdf/XGR-rdb2rdf/.
[15] C. Marie, G. Hallmark, and A. Paschke, "RIF Production Rule Dialect (Second Edition)", W3C Recommendation, **(2013)**, http://www.w3.org/TR/rif-prd/.
[16] M. Hert, G. Reif, and H. C. Gall, "A comparison of RDB-to-RDF mapping languages", Proceedings of the 7th International Conference on Semantic Systems, **(2011)**; Graz, Austria.
[17] J. Sequeda, S. H. Tirmizi, O. Corcho, and D. P. Miranker, "Survey of directly mapping SQL databases to the semantic web", Knowledge Engineering Review, vol. 26, no. 4, **(2011)**, pp. 445-486.
[18] C. Bizer and R. Cyganiak, "D2RQ — lessons learned," Position paper for the W3C Workshop on RDF Access to Relational Databases, **(2007)**, http://www.w3.org/2007/03/RdfRDB/papers/d2rq-position paper.

[19] V. Eisenberg, Y. Kanza, "D2RQ/update: updating relational data via virtual RDF", Proceedings of the 21st World Wide Web Conference (Companion Volume), **(2012)**; Lyon, France.
[20] F. Cerbah, "Learning highly structured semantic repositories from relational databases: the RDBToOnto tool", The Semantic Web: Research and Applications: 5th European Semantic Web Conference, **(2008)**; Tenerife, Canary Islands, Spain.
[21] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller, "Triplify: light-weight linked data publication from relational databases", Proceedings of the 18th International Conference on World Wide Web, Madrid, Spain, 20-24 April 2009, ACM **(2009)**, pp. 621-630.
[22] O. Erling and I. Mikhailov, "Virtuoso: RDF support in a native RDBMS," in Semantic Web Information Management, R. D. Virgilio, F. Giunchiglia, and L. Tanca, Eds., Springer Berlin Heidelberg **(2010)**, pp. 501-519.
[23] J. Sequeda and D. Miranker, "Ultrawrap: SPARQL execution on relational data," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 22, **(2013)**, pp. 19-39.
[24] E. Prud'Hommeaux, A. Seaborne, SPARQL query language for RDF, W3C Recommendation, **(2008)**, http://www.w3.org/TR/rdf-sparql-query.

Authors

Ying Chen received his B.S. degree in computer science and technology and M.S. degree in computer science from Southwest Jiaotong University, Chengdu, China in 2004 and 2007, respectively. Currently he is a Ph.D. candidate at College of Computer and Information in Hohai University, Nanjing, China. He is also working as a Lecturer in the Department of Computer Science and the Institute of Image Processing and Pattern Recognition at Taizhou University, Taizhou, China. His research interests include databases, the semantic web, and semantic multimedia annotation.

Zhuoming Xu received his B.S. and M.S. degrees in computer science from Hohai University, China, in 1986 and 1994, respectively, and the Ph.D. degree in computer science from Southeast University, China, in 2005. He was a visiting professor at Indiana University Bloomington, USA, from May 2011 to May 2012. He is currently a full professor (since 2004) of computer science at the College of Computer and Information in Hohai University, Nanjing, China, where he was the Chairman (2004–2006) of the Department of Computer Science and Engineering and the Deputy Director (2007–2012) of Science and Technology Office at Hohai University. Prof. Xu was a member of the council of China Computer Federation (CCF) from 2008 to 2011, and has been the Vice Chair of the Technical Committee on Computer Application of Jiangsu Computer Federation (JSCF) of China since 2010 to present. He has been involved, as principal investigators, in various research grants in the field of data management founded by the Chinese government. His research interests include databases, information retrieval, web data management, ontological engineering, and the semantic web.