

RUBYA: a Tool for Generating Rules for Incremental Maintenance of RDF Views

Vânia M. P. Vidal¹, Marco A. Casanova², Valéria M. Pequeno³, Narciso Arruda¹, Diego Sá¹, and José M. Monteiro¹

¹ Federal University of Ceará, Fortaleza, CE, Brazil
{vvidal, jmmfilho, narciso, diego}@lia.ufc.br

² Pontifical Catholic University of Rio de Janeiro, RJ, Brazil
casanova@inf.puc-rio.br

³ INESC-ID Porto Salvo, Portugal
vmp@inesc-id.pt

Abstract. We present Rubya, a tool that automatically generates the RDF view defined on top of relational data and all rules required for the incremental maintenance of the RDF view. Our approach relies on the designer to specify a mapping between the relational schema and a target ontology and results in a specification of how to represent relational schema concepts in terms of RDF classes and properties of the designers choice. Based on this mapping, the rules for incrementally maintenance of the RDF view are generated.

Keywords: RDF View Maintenance, RDB-to-RDF, Linked Data

1 Introduction

The Linked Data initiative [1] promotes the publication of previously isolated databases as interlinked RDF triple sets, creating a global scale dataspace known as the Web of Data. Since large volume of data is stored in relational data, making relational databases accessible to the Web of Data has special significance.

A general way to publish relational data in RDF format is to create RDF views of the relational data. The contents of views can be materialized to improve query performance and data availability. However, to be useful, a materialized view must be continuously maintained to reflect dynamic source updates.

In this demo, we show a framework named *RUBYA* (**R**ules **by** assertion), based on rules, for the incremental maintenance of external RDF views defined on top of relational data. Rubya has two main functionalities: 1) the generation of mappings between the relational schema and a target ontology; and 2) the generation of the rules required for the incremental maintenance of the view, based on the mapping initially generated. In Section 2, we further detail the *Rubya* tool.

The demo video is available at <http://tiny.cc/rubya>. First, the video shows, with the help of a real-world application, the process of defining the RDF view and generating the maintenance rules with *Rubya*. Then, it shows some practical

examples of using the rules for incremental maintenance of a materialized RDF view. For more information see <http://www.arida.ufc.br/ivmf/>.

2 Generating Rules with Rubya

Figure 1 depicts the main components of the framework. Briefly, the administrator of a relational database, using *Rubya*, should create RDF views and define a set of rules using *Rubya* - Figure 1(a). These rules are responsible for: (i) computing the view maintenance statements necessary to maintain a materialized view \mathbf{V} with respect to base updates; and (ii) sending the view maintenance statements to the *view controller* of \mathbf{V} - Figure 1(b). The rules can be implemented using triggers. Hence, no middleware system is required. The *view controller* for the RDF view has the following functionality: (i) receives the view maintenance updates from the RDB server and (ii) applies the updates to the view accordingly.

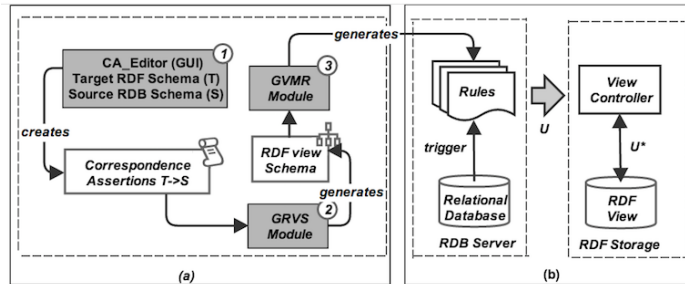


Fig. 1. Suggested Framework.

The process of defining the RDF view and generating the maintenance rules with *Rubya* consists of three steps:

STEP 1 (Mapping specification): Using the correspondence assertions editor of *Rubya*, the user loads the source and target schema and then he can draw Correspondence Assertions (CAs) to specify the mapping between the target RDF schema and the source relational schema. The demo video shows how the *CA_Editor* helps the user graphically to define CAs.

A CA can be: (i) a class correspondence assertion (CCA), which matches a class and a relation schema; (ii) an object property correspondence assertion (OCA), which matches an object property with paths (list of foreign keys) of a relation schema; or (iii) a datatype property correspondence assertion (DCA), which matches a datatype property with attributes or paths of a relation schema. CAs have a simple syntax and semantics and yet suffice to capture most of the subtleties of mapping relational schemas into RDF schemas. Figure 2 shows some examples of correspondence assertions between the relational schema *ISWC_REL*

and the ontology *CONF_OWL*. CCA1 matches the class *foaf:Person* with the relation *Persons*. We refer the reader to [7] for the details and motivation of the mapping formalism.

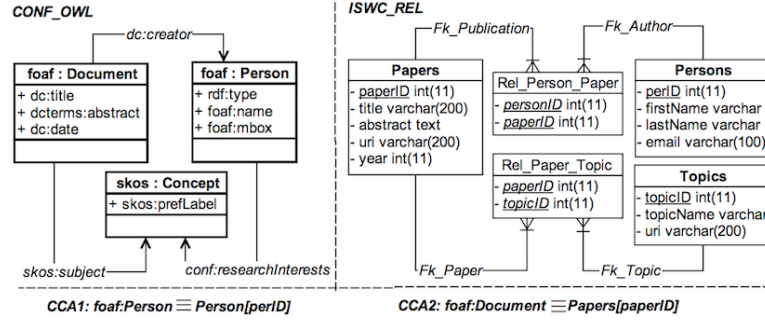


Fig. 2. *CONF_OWL* and *ISWC_REL* schemas and some examples of CAs.

STEP 2 (RDF view creation): The GRVS module automatically generates the RDF view schema, which is induced by the CAs defined in Step 1. The vocabulary of the RDF view schema contains all the elements of the target RDF schema that match an element of the source relational schema.

STEP 3 (Rule generation): The GVMR module automatically generates the set of rules required to maintain the RDF view defined in Step 2. The process of generating the rules for a view \mathbf{V} consists of the following steps: (a) Obtain, based on the CAs of \mathbf{V} , the set of all relations in the relational schema that are relevant to \mathbf{V} . (b) For each relation R that is relevant to \mathbf{V} , three rules are generated to account for insertions, deletions and updates on R .

Two procedures, *GVU_INSERTonR* and *GVU_DELETEonR*, are automatically generated, at view definition time, based on the CAs of \mathbf{V} that are relevant to R . Note that an update is treated as a deletion followed by an insertion, as usual. *GVU_INSERTonR* takes as input a tuple r_{new} inserted in R and returns the updates necessary to maintain the view \mathbf{V} . *GVU_DELETEonR* takes as input a tuple r_{old} deleted from R and returns the updates necessary to maintain the view \mathbf{V} . In [7], we present the algorithms that compile *GVU_INSERTonR* and *GVU_DELETEonR* based on the CAs of \mathbf{V} that are relevant to R .

Once the rules are created, they are used to incrementally maintain the materialized RDF view. For example, Figure 3 shows the process to update a RDF view when an insertion occurs on the relation *Papers*. When an insertion occurs on *Papers*, a corresponding trigger is fired. The trigger computes the view maintenance statements U , and sends it to the view controller. The view controller computes the view updates U^* , and applies it to the view state.

Rubya was developed in Java and relational data, triggers and stored procedures are stored using an Oracle database. The materialized RDF view is published using Fuseki. From Fuseki, the RDF triples can be updates through

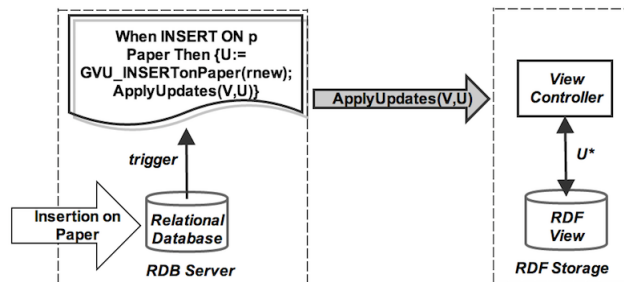


Fig. 3. Using the rules generated by *Rubya* when insertions occurs on Papers.

SPARQL update and they can be queried through a SPARQL endpoint. We use Jena [2] for communication with Fuseki. It is important to note that we can easily adapt *Rubya* to use other relational database than Oracle (since it support trigger, stored procedures and Java) and other RDF store.

3 Related Work

There is significant work on reusing relational data in terms of RDF (see a survey in [5,6]). Karma [4], for example, is a tool to semi-automatically create mapping from a source to a target ontology. In our tool, the user defines mappings between a source and a target ontology using a GUI. The novelty of our proposal is that we generate rules to maintain the RDF views.

[3] proposes the R3M tool in order to maintain RDF views over RDF databases. In that work, there is not any difference between the structure of the view and the data source, i.e., the RDF view contains a subset of classes and properties of the data source. In this case, direct mappings are used to match the elements of the view and the data source and the process of the view maintenance is very simple. Our approach differs from [3] mainly in two aspects: 1) our data source are relational databases, while the above papers is RDF databases; 2) our tool deals with more complex types of mappings. For example, our tool can deal with situations such as, in the relational database the attributes *firstName* and *lastName* be mapped to the property *foaf:name*. We also can deal with mappings when the RDF view describes a property which the data source does not include, although the source can express it through a path which it possess. For example, consider the schemas in Figure 2. The property *skos:subject* can be mapped to the path of Papers that includes the foreign keys *fk_Paper* and *fk_Topics*. For these cases involving complex mappings, simple view maintenance approaches are not a solution, and other approaches are necessary in order to correctly translate updates from the data source into updates in the RDF view.

4 Conclusions

In this paper, we present *Rubya*, a tool for incremental maintenance of external RDF views defined on top of relational data. More details about the theoretical fundamentals and algorithms used in Rubya can be found in [7]. A preliminary version of Rubya was presented, as poster, in [8].

Our approach is very effective for an externally maintained view because: the view maintenance rules are defined at view definition time; no access to the materialized view is required to compute the view maintenance statements propagated by the rules; and the application of the view maintenance statements by the *view controller* does not require any additional queries over the data source to maintain the view. This is important when the view is maintained externally [7], because accessing a remote data source may be too slow.

The use of rules is therefore an effective solution for the incremental maintenance of external views. However, creating rules that correctly maintain an RDF view can be a complex process, which calls for tools that automate the rule generation process, as Rubya does.

Acknowledgments. This work was partly funded by FCT with references UID/CEC/50021/2013 and EXCL/EEI-ESS/0257/2012 (DataStorm) and grants SFRH/BPD/76024/2011; by CNPq, under grants 442338/2014-7 and 303332/2013-1; and by FAPERJ, under grant E-26/201.337/2014.

References

1. Berners-Lee, T.: Design issues: Linked data. 2006. Available in: <http://www.w3.org/DesignIssues/LinkedData.html>
2. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the semantic web recommendations. In: WWW Alt.'04, pp. 74–83. ACM, New York, NY, USA (2004)
3. Deng, Y., Hung, E., Subrahmanian, V.: Maintaining rdf views. Tech. Rep. Tech. Rep. CS-TR-4612 (UMIACS-TR-2004-54), University of Maryland Institute for Advanced Computer Studies (2004)
4. Knoblock, C.A., et al.: Semi-automatically Mapping Structured Sources into the Semantic Web. In: ESWC, pp. 375–390. Springer-Verlag, Berlin, Heidelberg (2012)
5. Michel, F., Montagnat, J., Faron-Zucker, C.: A survey of RDB to RDF translation approaches and tools. Research report, I3S (2014)
6. Spanos, D.E., Stavrou, P., Mitrou, N.: Bringing Relational Databases into the Semantic Web: A Survey. *Semantic Web Journal* **3**(2), 169–209 (2012)
7. Vidal, V.M.P., Casanova, M.A., Cardoso, D.S.: Incremental Maintenance of RDF Views of Relational Data. In: OTM 2013 Conferences, pp. 572–587. Austria (2013)
8. Vidal, V.M.P., Casanova, M.A., Monteiro, J.M., Jr., N.M.A., Cardoso, D.S., Pequeno, V.M.: A framework for incremental maintenance of RDF views of relational data. In: ISWC 2014 Posters & Demonstrations Track, pp. 321–324 (2014)