A Framework for Incremental Maintenance of RDF Views of Relational Data

Vânia M. P. Vidal[1], Marco A. Casanova[2], José M. Monteiro[1], Narciso Arruda[1], Diego Sá[1], and Valéria M. Pequeno[3]

[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] DMIR, INESC-ID Porto Salvo, Portugal
vmp@inesc-id.pt

Abstract. A general and flexible way to publish relational data in RDF format is to create RDF views of the underlying relational data. In this paper, we demonstrate a framework, based on rules, for the incremental maintenance of RDF views defined on top of relational data. We also demonstrate a tool that automatically generates, based on the mapping between the relational schema and a target ontology, the RDF view exported from the relational data source and all rules required for the incremental maintenance of the RDF view.

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, thereby creating a global scale dataspace, known as the Web of Data. However, the full potential of linked data depends on how easy it is to publish data stored in relational databases (RDBs) in RDF format. This process is often called RDB-to-RDF.

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, based on rules, for the incremental maintenance of external RDF views defined on top of relational data. Figure 1 depicts the main components of the framework. Briefly, the administrator of a relational data-base, using *Rubya* (**R**ules **by a**ssertion), 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 **V** with respect to base updates; and (ii) sending the view maintenance statements to the *view controller* of **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.

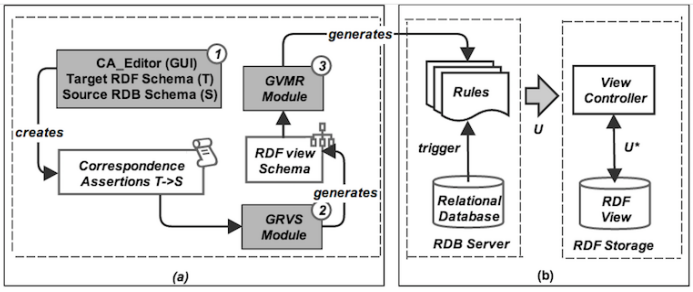

Fig. 1. Suggested Framework.

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 [4], 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. In Section 2, we further detail the *Rubya* tool that, based on the mapping between the relational schema and a target ontology, automatically generates the RDF view exported from the relational data source and the set of rules required for the incremental maintenance of the RDF view.

The demo video is available at *http://tiny.cc/rubya*. First, the video shows, with the help of a real-word 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 highlights the main components of *Rubya*. 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 [4, 5] for the details and motivation of the mapping formalism.

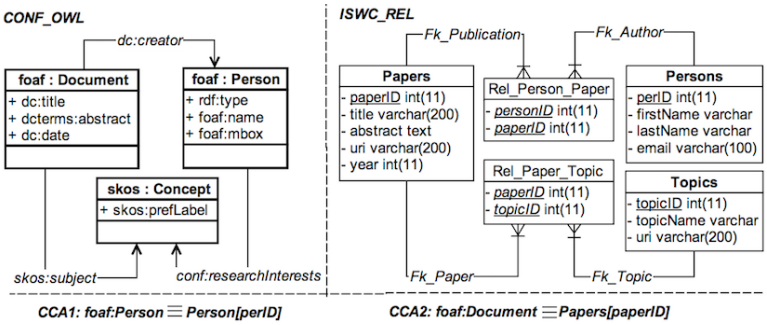

Fig. 2. *CONF_OWL and ISWC_REL schemas and some examples of CAs.*

STEP 2 (RDF view creation): The GRVS module automatically generates the RDF view schema, which is induced by the correspondence assertions 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 **V** consists of the following steps: (a) Obtain, based on the CAs of **V**, the set of all relations in the relational schema that are relevant to **V**. (b) For each relation R that is relevant to **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 **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 **V**. *GVU_DELETEonR* takes as input a tuple r_{old} deleted from R and returns the updates necessary to maintain

the view **V**. In [4], we present the algorithms that compile *GVU_INSERTonR* and *GVU_DELETEonR* based on the CAs of **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 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.

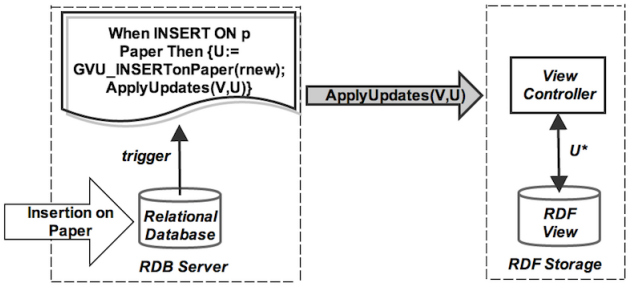

Fig. 3. Using the rules generated by *Rubya* when insertions occurs on Papers.

3 Conclusions

In this paper, we present *Rubya*, a tool for incremental maintenance of external RDF views defined on top of relational data. There is significant work on reusing relational data in terms of RDF (see a survey in [3]). Karma [2], 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.

References

1. Berners-lee, T., design issues: Linked data, `http://www.w3.org/DesignIssues/LinkedData.html`
2. Knoblock, C.A., et al.: Semi-automatically Mapping Structured Sources into the Semantic Web. In: ESWC, pp. 375–390. Springer-Verlag, Berlin, Heidelberg (2012)
3. Spanos, D.E., Stavrou, P., Mitrou, N.: Bringing Relational Databases into the Semantic Web: A Survey. Semantic Web Journal **3**(2), 169–209 (2012)
4. 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)
5. Vidal, V.M.P., Casanova, M.A., Neto, L.E.T., Monteiro, J.M.: A Semi-Automatic Approach for Generating Customized R2RML Mappings. In: SAC, pp. 316–322 (2014)