CEUR-WS.org/Vol-2456/paper56.pdf

ShERML: Mapping Relational Data to RDF*

Iovka Boneva, Jose Lozano, and Sławek Staworko

Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France

Abstract. We present ShERML, a tool for facilitating the data exchange from relational data to RDF. The main feature of the tool is a graphical user interface for designing a relational-to-RDF mapping by drawing arrows between schematic representations of the relational schema on the one hand, and a SHACL or ShEx schema on the other hand. The mapping thus constructed can be used directly by the tool for materializing the RDF graph, or be exported as an R2RML mapping.

Keywords: RDF, ShEx, SHACL, Graphical Mapping Language, R2RML

1 Introduction

RDF and Semantic Web data often originate from pre-existing relational databases. The problem to export relational data to RDF is known as relational to RDF data exchange, and can be solved either by writing ad-hoc export scripts, or more recently by using dedicated declarative languages such as the W3C standard R2RML. The latter allows to define *mappings* that use logical tables (typically SQL queries) and how each row of such table is used to construct a number of RDF triples. There exist a number of mapping editors to facilitate the creation of R2RML mappings [5,3,4,1].

ShERML is inspired by the well known relational-to-relational data exchange tool Clio[2], in which mappings are defined by drawing arrows between the source and the target schema, as shown in Fig. 1. In ShERML we use Shape Constraint Languge (SHACL) or Shape Expressions Language (ShEx) as target schemas. Both languages were specifically designed to describe the vocabulary and the desired structure of an RDF graph. They use *shapes* to describe how a particular piece of information should be structured. Collections of shapes with references to each other are called *shapes schemas*. Using a shapes schema for relational to RDF mappings presents several advantages. On the one hand, fixing the target vocabulary prior to the design of the mapping allows to separate concerns. On the other hand, compared to ontologies, shapes schemas are more flexible as they allow to define (1) a target vocabulary mixing terms from different ontologies, and (2) structural requirements on the graph fitted for a particular existing application. Additionally, ShERML checks whether the mapping defined by the user would generate an RDF graph compliant with the given shapes schema.

*Copyright © 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

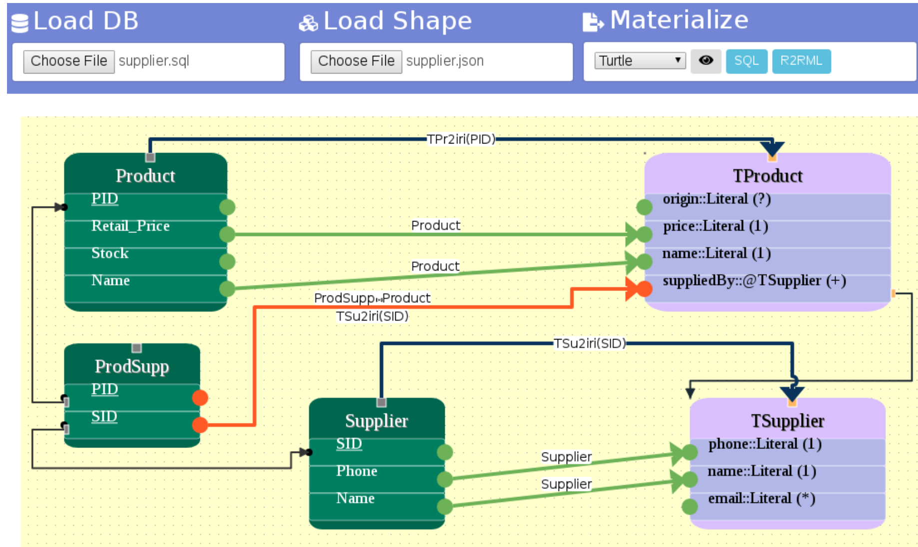

Fig. 1. A mapping from a relational database to shapes schema defined with ShERML.

2 Architecture and Implementation

The architecture of ShERML is presented in Fig. 2. The core element of ShERML is the *Graphical Mapping Language* (GML), which allows to define mappings from relational databases to RDF while specifying how the results of SQL queries should be organized to fit the shapes of a shapes schema. ShERML provides a *Graphical Mapping Language Editor* that allows to define GML mappings while allowing a high degree of customization of the defined mapping. A GML mapping can be fed into the *Materializer* that executes it by running SQL queries on the relational database and constructing the desired RDF graph. Additionally, the GML script can be translated to a corresponding R2RML script with the help of the *Converter*. The online video demo, source code and installation instructions are available on this repository `https://github.com/josemachino/ShERML`.

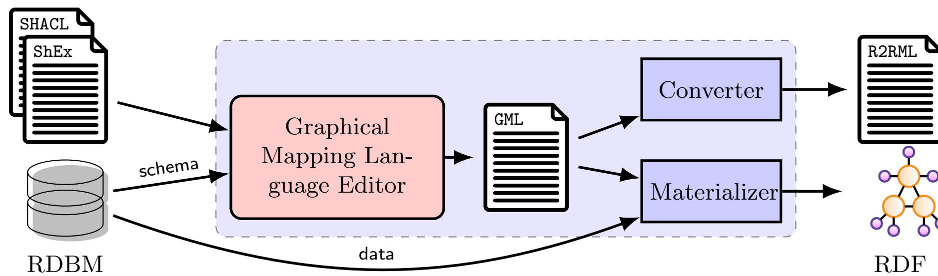

Fig. 2. ShERML Architecture and Workflow.

3 Demo Scenario

We demonstrate the principal functionality of ShERML with the following use case. We shall export the contents of a relational database with products and their suppliers as seen on the left side in Fig. 1. The database consists of three tables: *Product* with the information on products, *Supplier* with the information on suppliers, and *ProdSupp* with information on what supplier provides which product. The data is to be exported to RDF whose structure is defined with the schema present on the right side in Fig. 1. In essence, it defines two types of RDF nodes: TProduct for products and TSupplier for suppliers. The type TProduct requires a node to have a single name, a single price, and an optional origin information. Additionally, a product node must have one or more suppliers. A supplier node must have a name, a phone number, and an arbitrary number of email addresses. During the demonstration, we shall construct the desired GML mapping, in Fig. 1, with the following steps:

Loading schema. Before defining the mapping, both schemas need to be loaded from their source files. Each table and each shape are represented with a single box, each containing the list of attributes/property names. Each element of the list has a dedicated circular *anchor*, for creating connections later on. Anchors use two colors: *orange* anchors identify elements that reference other elements i.e., foreign keys in the relational schema and shape references in shapes schema; *green* anchors identify remaining elements that store values.

Defining mapping. The mapping is defined with the convenience of drawing arrows from the elements of relational schema to the elements of the shape schema. There are three kinds of arrows:

IRI arrow → connects relation to the corresponding shape (it does not use anchors). It defines how rows from a table are to be mapped to IRI nodes satisfying a given shape constraint. For instance, table *Product* is mapped to the shape TProduct and for every row in *Product* a corresponding IRI is created in the output RDF using a specified IRI constructor `TPr2iri` applied to the key attribute *PID* (more details on IRI constructors below).

Property arrow •→• connects relation attributes to literal properties of the shapes; it can only connect two green anchors. It defines how table rows are mapped to triples. A property arrow is dependent on an IRI arrow that defines the IRI nodes for which the property values are generated. For instance, the arrow from *Product.Name* to TProduct.name maps every row in *Product* to a triple whose subject is `TPr2iri`(*PID*), the predicate is name, and the object is the value of the *Name* attribute in the row.

Reference arrow •→• connects a foreign key of a table to a reference property in the shape schema; it can only connect two orange anchors. Similarly to a property arrow, a reference arrow is dependent on an IRI arrow and yields RDF triples, but this time the object is also an IRI rather than a literal. For instance, the arrow from *ProdSupp.SID* to TProduct.suppliedBy generates the triples that connect every product with each of its suppliers.

Both property and reference arrows are dependent on an IRI arrow that specifies the subject of the generated triples. The IRI arrow does not need to originate from the same table, and those arrows are annotated with the join expressions needed to reach the relevant IRI arrow cf., the expression $\mathit{ProdSupp} \bowtie \mathit{Product}$ on the arrow from $\mathit{ProdSupp}.\mathit{SID}$ to TProduct.suppliedBy.

Both IRI and reference arrows are annotated with IRI constructors that specify how the IRI nodes are to be obtained. By default when loading a shape schema, the system constructs default IRI constructors for every shape in a manner consistent with how IRIs are constructed in R2RML. For instance, for the TProduct shape, whose full IRI is `http://inria.fr/TProduct`, the default IRI constructor `TPr2iri` basically appends the given value to this IRI e.g., `TPr2iri(P1)` returns `http://inria.fr/TProduct/P1`.

Refining GML mapping. The mapping can then be closely inspected and refined further. For this purpose ShERML offers a visual inspector that reorganizes the GML by grouping the arrows by the target shapes. It allows adding filter conditions and, in our demonstration we shall add a condition to the TProduct IRI arrow to export only products whose stock is lower than 1000.

RDF materialization and R2RML export. Once the GML mapping is defined, ShERML can export the contents of the relational database to RDF in a chosen format (Turtle, N-triples, or RDF/JSON). Finally, the system allows us to convert the GML mapping to an R2RML mapping that can then be used by third-party applications.

During the demonstration we shall also present static and runtime consistency checking and property value transformations.

4 Conclusions

We have presented a demo of ShERML, a versatile system for defining mappings from relational databases to RDF with target shape schema using an intuitive and powerful graphical mapping language. As next step we plan to use the tool for expressing existing relational to RDF data exchange scenarios currently written as python programs. This will allow to prioritise the future extensions of ShERML such as extending its mapping language or its GUI.

Acknowledgments This work was partially supported by a grant from CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020 and by the ANR project DataCert ANR-15-CE39-0009.

References

1. Crotti Junior, A., Debruyne, C., O'Sullivan, D.: Juma: An editor that uses a block metaphor to facilitate the creation and editing of R2RML mappings. In: ESWC 2017
2. Fagin, R., Haas, L.M., Hernández, M., Miller, R.J., Popa, L., Velegrakis, Y.: Clio: Schema Mapping Creation and Data Exchange, pp. 198–236. Springer Berlin Heidelberg (2009)
3. Heyvaert, P., Dimou, A., Herregodts, A.L., Verborgh, R., Schuurman, D., Mannens, E., Van de Walle, R.: RMLEditor: a graph-based mapping editor for Linked Data mappings. In: ESWC 2016
4. Rml.x visual editor. `http://pebbie.org/mashup/rml`
5. Sengupta, K., Haase, P., Schmidt, M., Hitzler, P.: Editing R2RML mappings made easy. In: ISWC 2013