Simplified RDB2RDF Mapping

Claus Stadler[*]
Department of Computer Science, University of Leipzig, Germany
cstadler@informatik.uni-leipzig.de

Jörg Unbehauen
Department of Computer Science, University of Leipzig, Germany
unbehauen@informatik.uni-leipzig.de

Patrick Westphal
Department of Computer Science, University of Leipzig, Germany
pwestphal@informatik.uni-leipzig.de

Mohamed Sherif
Department of Computer Science, University of Leipzig, Germany
sherif@informatik.uni-leipzig.de

Jens Lehmann
Department of Computer Science, University of Leipzig, Germany
lehmann@informatik.uni-leipzig.de

ABSTRACT

The combination of the advantages of widely used relational databases and semantic technologies has attracted significant research over the past decade. In particular, mapping languages for the conversion of databases to RDF knowledge bases have been developed and standardized in the form of R2RML. In this article, we first review those mapping languages and then devise work towards a unified formal model for them. Based on this, we present the Sparqlification Mapping Language (SML), which provides an intuitive way to declare mappings based on SQL VIEWS and SPARQL construct queries. We show that SML has the same expressivity as R2RML by enumerating the language features and show the correspondences, and we outline how one syntax can be converted into the other. A conducted user study for this paper juxtaposing SML and R2RML provides evidence that SML is a more compact syntax which is easier to understand and read and thus lowers the barrier to offer SPARQL access to relational databases.

1. INTRODUCTION

Despite the success of semantic technologies, a large share of structured knowledge still resides in relational databases. For this reason, significant research effort has been invested by the Semantic Web community into making relational databases available as RDF.

Due to the strong interest in this area, several approaches and languages for mapping relational data to triples have been devised, in particular the W3C standard R2RML[1]. While having a standard itself is of high importance, we argue that R2RML has some drawbacks on the syntactical level: Writing RDF views in R2RML is very verbose and arguably not as intuitive as it could be. The choice of using RDF as base syntax for R2RML has the advantage that people writing mappings can be expected to know RDF. However, there is a significant gap between the relational database structure and the structure of the R2RML mapping specifications. While graphical editors, such as [11][7], partially mitigate the problem of having to write those mapping definitions, these also have their limitations. In particular, they would have to support both, the full feature set of the mapping language while still be efficient to work with and producing human readable output. Moreover, in some environments, Web based editors in the spirit of phpmyadmin[2] may pose security risks or are not convenient, since many database and RDF experts are simply used to work on text files and textual representations of data and queries. While they appreciate unobtrusive help, like syntax checking or code completion, a graphical user interface might impose an unfitting work flow, for example when an administrator is used to be able to perform small database related tasks via a command line interface. In this work, we introduce the Sparqlification Mapping Language (SML) as a human friendly alternative to R2RML. It is noteworthy, that non-RDF syntaxes for which RDF-based versions exist are commonly used the Semantic Web. For example, while e.g. OWL ontologies can be written directly in RDF, Manchester OWL Syntax[3] is a popular and concise alternative used in the primer of the OWL 2 specification itself. As another example, while SPARQL queries in principle can be written in RDF using the SPIN SPARQL Syntax[4], it is uncommon to do so unless special use cases demand this.

SML is based on work towards a unified formal model for RDB2RDF mappings. While it has equal expressiveness to R2RML, it uses a different syntactical approach: It blends the traditional SQL `CREATE VIEW` statements with SPARQL `CONSTRUCT` queries. Both features can be expected to be fa-

[*] Corresponding Author

[1] `http://www.w3.org/TR/r2rml/`

[2] `http://www.phpmyadmin.net`

[3] `http://www.w3.org/TR/owl2-manchester-syntax/`

[4] `http://spinrdf.org/sp.html`

WWW2015 Workshop: Linked Data on the Web (LDOW2015).

miliar to persons working on RDB2RDF data integration and combined provide a more concise syntax than R2RML. In fact, we believe that for RDF itself, history has shown that the seemingly obvious choice of syntactically building on XML has had several drawbacks and the special purpose language Turtle meanwhile enjoys high popularity for manually crafting and editing RDF documents and Turtle 1.1 became a W3C Proposed Recommendation in 2014. Similarly, we believe a more intuitive special purpose RDB2RDF mapping language can provide similar benefits.

The research on the syntax of SML builds on a comparison of RDB2RDF mapping languages and a subsequently defined formal model of those languages. Languages like R2RML and SML are syntactic instances of this formal model. We use this model to highlight the equivalence between the languages and derive approaches for converting between them. In particular, this implies that any processor, which can work on the W3C R2RML standard, can also use SML as input and no further implementation effort is required to use SML in combination with a number of RDB2RDF engines. Our main argument is that SML despite its simplicity provides equal expressiveness and is, therefore, a viable alternative to R2RML. The contributions of the article are as follows:

- Definition of the compact SML mapping language with equal expressiveness to R2RML
- Comparison of RDB2RDF mapping languages.
- A unified formal model of RDB2RDF mapping languages.
- Converters from R2RML to SML and vice versa.
- Syntax highlighting definition for the editor *vim* and an online SML editor with syntax and error highlighting as a demonstrator. Although this component is an engineering effort, it contributed to the fairness of the user study in terms of providing comparable tool support for both R2RML and SML.

All tools, demos and the specification of the SML syntax, are available at `http://sml.aksw.org`.

The remainder of the article is structured as follows: In Section 2 we review existing RDB2RDF mapping languages. Subsequently, in Section 3 we present a corresponding formal model. The SML syntax is introduced in Section 4, whereas Section 5 compares it to R2RML. In Section 6 the conversion approach from SML to R2RML is described. In Section 7, we describe a user study via a public survey with 46 participants amounting to almost 16 hours of survey completion time. Finally, we conclude this paper in Section 8.

2. RDB2RDF SYSTEMS AND MAPPING LANGUAGES REVIEW

The mapping of *relational databases* (RDB) to the *Resource Description Framework* (RDF) is of keen interest from the inception of the *Semantic Web* as exemplified in [6]. The exposition of such previously constrained data allows integration and interlinking with other data on the Web. Based on this need, multiple tools and approaches emerged. In an approach for fostering interoperability between those tools, the standardization of the RDB2RDF Mapping Language (R2RML) was initiated by the W3C RDB2RDF working group[5].

R2RML is defined in [4] as a mapping language for describing customized mappings of relational data into RDF. The R2RML specification is accompanied by the Direct Mapping (DM) specification [2], describing a standard way of translating a relational database into RDF without the use of a customized mapping definition. An R2RML mapping definition is represented in RDF using the R2RML vocabulary and serialized in the Terse RDF Triple Language (Turtle). It can be used to either store converted relational data in an RDF dump file, expose the data as Linked Data or allow querying it via a SPARQL endpoint. A more general overview of mapping tools for structured sources is given in [14]. Recent efforts, such as [5], propose extensions of R2RML for non-relational sources by adding support for the use of e.g. XPath[6] and JSONPath expressions in the mappings. In this work we focus on relational data.

With the advent of R2RML, vendors took up the standard and either modified their existing tools to additionally support R2RML or created tools fully based on the standard. In general these tools can be categorized with regards to different dimensions, with the type of data exposition and the mode of querying the underlying database being the most distinctive. A list of existing R2RML tools is given in Table 1. These tools have in common that they all allow the exposition as SPARQL endpoint and all employ SPARQL-to-SQL translation.

The R2RML tools use the mapping definition expressed in R2RML to connect the relational data with a domain ontology. The domain ontology describes the actual RDF data exposed and consists of standard vocabularies and custom created terms depending on the use case. Listing 1 provides an example of an R2RML mapping of a simple employee table only containing IDs (`EMPNO`) and names (`ENAME`).

```
<#TriplesMap1>
    rr:logicalTable [ rr:tableName "EMP" ];
    rr:subjectMap [
        rr:template "http://data.example.com/employee
            /{EMPNO}";
        rr:class ex:Employee;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:name;
        rr:objectMap [ rr:column "ENAME" ];
    ].
```

Listing 1: Example of an R2RML mapping.

D2RQ-ML[7] is another declarative language for mapping RDB to RDF, supported by the D2R server. As D2R is one of the most popular RDB2RDF solutions, its mapping language is also supported by other tools like UltraWrap. The D2RQ mapping itself is an RDF document as well, usually written in Turtle syntax. The mapping defines a virtual RDF graph that contains information from the database. This is similar to the concept of views in SQL, except that the virtual data structure is an RDF graph instead of a virtual relational table. The *D2RQ Platform* provides SPARQL access, a Linked Data server, an RDF dump generator, a simple HTML interface, and Jena API access to D2RQ-

[5] `http://www.w3.org/2001/sw/rdb2rdf/`

[6] `http://www.w3.org/TR/xpath-30/`

[7] `http://d2rq.org/d2rq-language`

mapped databases. Listing 2 shows an example of a D2RQ mapping from a conferences table in a database to the conference class in an ontology.

```
map:Database1 a d2rq:Database;
    d2rq:jdbcDSN "jdbc:mysql://localhost/iswc";
    d2rq:jdbcDriver "com.mysql.jdbc.Driver";
    d2rq:username "user";
    d2rq:password "password";
    .
map:Conference a d2rq:ClassMap;
    d2rq:dataStorage map:Database1.
    d2rq:class :Conference;
    d2rq:uriPattern "http://conferences.org/comp/confno@@Conferences.ConfID@@";
    .
map:eventTitle a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Conference;
    d2rq:property :eventTitle;
    d2rq:column "Conferences.Name";
    d2rq:datatype xsd:string;
        .
```

Listing 2: D2RQ map for conferences.

Generally speaking, D2RQ-ML is close to R2RML with some notable distinctions. D2RQ-ML includes the database connection information in the mapping file and uses different constructs to express joins between tables.

Another notable approach is utilized in the ontop[9] platform by the *Knowledge Representation meets Databases (KRDB)*[8] research group. Ontop supports mapping definitions in its own language and R2RML. Quest, the SPARQL engine/reasoner in ontop, implements query rewriting techniques that translate SPARQL into SQL. Listing 3 shows an example from the ontop documentation[9].

```
[MappingDeclaration] @collection [[
  mappingId     Book collection
  target        :BID_{id} a :Book .
  source        SELECT id FROM books
]]
```

Listing 3: Example of the Ontop mapping language

Virtuoso RDF Views [1] is another tool specific mapping language. It is part of OpenLink's Virtuoso Universal Server[10]. Virtuoso RDF Views provide a declarative Meta Schema Language for mapping of SQL data to RDF ontologies and preceded Virtuoso's R2RML support. The corresponding mappings are dynamic, such that changes to the underlying data are reflected immediately in the RDF views. OpenLink Virtuoso Universal Server includes SPARQL support and an RDF data store tightly integrated with its relational storage engine. An example of a Virtuoso RDF View definition is given in Listing 4.

```
graph <http://localhost/testdata/products#>
subject prd:product_iri(PRODUCT.PRODUCT_ID)
predicate rdf:type
object prd:Product
```

Listing 4: Virtuoso RDF views example

Besides the textual mapping languages, there are also tools providing a graphical representation of the mapping. The *Asio Semantic Web bridge* SBRD[11] or the more recent R2RML editor presented in [12] fall into this category.

SBRD utilizes *Snoggle*[12] for mapping from RDB to RDF. *Snoggle* is a graphical ontology mapper based on the *Semantic Web Rule Language* (SWRL)[13]. It allows users to draw ontologies and then create mappings between them on a graphical canvas. This mapping is then translated into SWRL/RDF or SWRL/XML.

An overview of the introduced RDB2RDF solutions is given in Table 1.

3. TOWARDS A UNIFIED FORMAL MODEL FOR RDB2RDF MAPPINGS

In this section, we outline a formal approach for mapping tabular data to RDF. For this purpose, we first briefly summarize fundamental concepts of the RDF data model. It should be noted, that we assume that RDF is generated by *row-wise* processing of the underlying relational data. Both R2RML and SML build on this assumption. Also, without loss of generality, we only consider quads rather than triples in the formalization. The reason is, that we can view any generated triple as being labeled with the URI of the graph it belongs to.

Preliminaries.

Let the RDF primitives be: $\mathcal{U}$ the set of URIs, $\mathcal{B}$ the set of blank nodes, $\mathcal{L}$ the set of literals and $\mathcal{V}$ the set of variables. Further:

- $\mathcal{T}$ is the set of all *RDF terms*, defined as $\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$.

Furthermore, we make use of the following notions:

- $\mathcal{J}$ is the joint set of RDF terms and variables, defined as $\mathcal{T} \cup \mathcal{V}$.
- $\mathcal{Q}$ is the set of all *quads*, defined as $\mathcal{J} \times \mathcal{J} \times \mathcal{J} \times \mathcal{J}$.
- A *quad pattern* Q is a finite, possibly empty, set of quads, defined as $Q \subset \mathcal{Q}$
- $\mathcal{R}$ is the set of all quad patterns, thus the powerset of $\mathcal{Q}$, denoted by $\mathcal{P}(\mathcal{Q})$
- A *quad* q is defined as $q \in \mathcal{Q}$.
- $vars(Q)$ is the set of variables appearing in Q.
- A *ground quad (pattern)* is a variable free quad (pattern).

Finally, we introduce our notion of a *relation instance* (short: relation) L, which, for convenience, we define as a *set of partial functions* that map attribute names to attribute values. It is noteworthy, that R2RML defines an entity referred to as *logical table*. Instances of this entity possess an *effective SQL query*[14] which can be evaluated over an instance of a database schema in order to obtain a relation.

Generating RDF from relations.

Based on the previously introduced primitives, we are now able to formally capture the nature of RDF mapping approaches for relational data.

A relational data to RDF (*R2R*) mapping m is a fourtuple (N, P, L, f):

[8] http://www.inf.unibz.it/krdb/

[9] https://github.com/ontop/ontop/wiki/ontopOBDAModel

[10] http://virtuoso.openlinksw.com/

[11] http://bbn.com/technology/knowledge/asio_sbrd

[12] http://bbn.com/technology/knowledge/snoggle

[13] http://www.w3.org/Submission/SWRL/

[14] http://www.w3.org/TR/r2rml/#dfn-effective-sql-query

Tool/Features	Mapping language	SPARQL Version	License	Support
Ontop [10]	Own language & R2RML	1.0	Free	Free
Revelytix Spyder	R2RML	1.1	Free	With fees
Asio SBRD	Graphical	—	Commercial	Commercial
Virtuoso RDF Views [1]	Own language & R2RML	1.1	Free	Free
D2RQ Platform [3]	D2RQ-ML	1.1	Free	Free
Morph [8]	R2RML	1.0	Free	Free
Ultrawrap [13]	R2RML	1.0	Commercial	Commercial
SparqlMap [15]	R2RML	1.0	Free	Commercial
Sparqlify	R2RML & SML	1.0	Free	Free

Table 1: Comparison between different mapping tools and languages.

- N is the *name* of a view.
- P is a *quad pattern* which acts as the *template* for the construction of triples and relating them to named graphs. The template is instantiated once for each row of the relation.
- L is a relation to be converted to RDF.
- f is a mapping with signature $L \to (\mathcal{V} \to \mathcal{T})$: f yields for each element of the relation L a partial function that binds the variables of the template P to RDF terms in $\mathcal{T}$. Note that it is not required for variables of P to be bound, which enables the support NULL values in the source data.

An R2R mapping is valid, if its *evaluation* yields an *RDF dataset*, as defined in the SPARQL secification[15].

Given a quad pattern $Q \subset \mathcal{Q}$ and a partial function $a : \mathcal{V} \to \mathcal{T}$, we define the substitution operator

$$\rho_{[a]} : \mathcal{R} \to \mathcal{R}$$

$\rho_{[a]}(Q)$ yields a new ground quad pattern Q' with all variables replaced in accordance with a. Any quads of Q with unbound variables in a are omitted in Q'.

An evaluation of a mapping m proceeds by passing each row of L as an argument to f, thereby obtaining the bindings for $vars(P)$, which are used to instantiate the template P for finally creating ground quads. Let $\mathcal{M}$ be the set of all mappings, then a function $eval : \mathcal{M} \to \mathcal{R}$ can be defined as:

$$eval(m) = \bigcup_{l \in L} \{\rho_{[f(l)]}(P)\} \qquad \text{with } m = (N, P, L, f)$$

What remains is to define a representation of the function f in terms of expressions. We refer to such a set of expressions as a *variable definition*. An analysis of the mapping languages revealed, that there is a small set of essential operations for RDB-to-RDF mappings, for which we devised an Extended Backus–Naur Form (EBNF) of an expression grammar as shown in Listing 5 and explained as follows.

In a first step, we need to be able to construct RDF terms from the underlying relation, hence we introduce the *rdf-term-ctor-expr*[16] production. Note that our `plainLiteral` and `typedLiteral` functions roughly correspond to the functions`STRDT` and `STRLANG` of the SPARQL standard, although in SML arguments may be of types other than string, such as when mapping a column of type real to a corresponding typed literal. Yet, in the future aliases may be introduced to SML for better alignment with existing SPARQL features.

An analysis of the mapping languages revealed, that there is a small set of essential operations for RDB-to-RDF mappings, namely *concat*, *str* and *urlEncode* and *percentEncode*[17]. These function symbols are usually used for the construction of URIs and IRIs from values of the underlying relation: The function symbol *concat* may be used to prepend a prefix IRI to one or more ID columns. The function symbol *str* corresponds to an implicit conversion and therefore usually does not have to be stated explicitly as it can be implied. It is needed to preserve type consistency: For instance, *concat* is only defined for string arguments. Therefore, *concat*('http://ex.org/', 1) would yield a type error without the prior conversion of the second argument to string.

Note, that although these functions could be applied in the underlying RDBMS, support at the mapping level opens possibilities for basic optimizations without the need to parse the involved SQL.

```
varDefinition = (var '=' rdf-term-ctor-expr)* ;

rdf-term-ctor-expr
    = bNode '(' expr ')'
    | uri '(' expr ')'
    | plainLiteral '(' expr (',' expr)? ')'
    | typedLiteral '(' expr ',' expr ')'
    ;

expr-list
    = (expr (',' expr)*)?
    ;

expr
    = var      // Denotes a reference to a column
    | str '(' expr ')'
    | concat '(' expr-list ')'
    | urlEncode '(' expr ')'
    ;
```

Listing 5: EBNF for variable definition expressions

Example: Assume a given relation holding the label of a product:

$$\{\{(\text{id}, 1), (\text{label}, \text{“Coke”})\}, \ldots\}$$

Assume that we aim to obtain the following assignment from variables to RDF terms:

$$\{\{(?s, \text{<http://ex.org/1>}), (?l, \text{“Coke”@en})\}, \ldots\}$$

Then a definition of f as

$$f : [\{?s = \text{uri}(\text{concat}(\text{'http://ex.org/'}, \text{str}(?id))), \\ ?l = \text{plainLiteral}(?label, \text{'en'})\}] \tag{1}$$

would yield the desired output.

[15] http://www.w3.org/TR/sparql11-query/#rdfDataset

[16] We use *ctor* as abbreviation for *constructor*

[17] http://tools.ietf.org/html/rfc3986

4. SML SYNTAX

In this section, we give an introduction to the SML syntax. The left hand side of Figure 1 shows an example of SML, whose syntactic constituents are explained as follows.

Recall that in the previous Section 3 we formally defined an R2R view as a four-tuple (N, P, L, f). The core syntax of an SML view definition comprises four parts that correspond directly to the formal definition. Additionally, SML features an optional *constraint* component for improving query performance. An SML view definition is composed of the following parts:

- The *name* of the view. This corresponds to an element of the set N.

- A *construct* clause, which consists of triple patterns which can be optionally associated with a specific named graph by surrounding them with `GRAPH` $G \{ \dots \}$, where G can be a variable name or an IRI. Hence, the syntax is equivalent to the *quads* production rule of the SPARQL 1.1 standard[18]. This corresponds to an element of P.

- A *FROM* clause, where a reference to a *logical table* can be specified. As in R2RML, this can be either an SQL SELECT statement, the name of a physical table or the name of a view. The former needs to be escaped in triple double-quotes, i.e. `"""SELECT ..."""`. The execution of a logical table's effective SQL query over an SQL connection yields a result set which formally corresponds to L.

- The *variable definition clause* acts as the bridge between the RDF and SQL data models, and is used to specify the creation of RDF terms from rows of the relation. It consists of a set of *variable definition statements* of the form *?var = rdf-term-ctor(expr*$_0$*, ..., expr*$_n$*)*, and should at least support the grammar defined in 5.

- Finally, there is a *CONSTRAINT* clause for specifying contstraints about variables on the RDF level. As such, it has no direct influence on the virtual RDF relation, but rather on query performance. The example in Figure 1 shows, that solely based on the definition of *?s = uri(?website)* we have no information about the set of URIs being created. Specifying such a type of constraints enables SPARQL-to-SQL rewriters, for instance, to prune joins whose join condition equates variables with disjoint sets of prefixes. Syntactically, up to now only stating prefix constraints is supported.

5. COMPARISON OF SML WITH R2RML

In this section, we summarize essential features of R2RML and explain how they relate to those of SML. R2RML mappings are expressed as RDF graphs for which R2RML by convention uses Turtle serialization. The fundamental class is `rr:TriplesMap`, whose instances are specifications of the triples to generate from an underlying logical table. As such, an instance of a TriplesMap corresponds to an SML view definition. In the following, we explain the most important attributes that TriplesMaps may have, and compare them to SML. Figure 1 shows a side-by-side comparison of the mapping languages for a specific example. Both syntactic formats can be converted to each other.

Defining Logical Tables.

R2RML defines the predicate `rr:logicalTable` to relate a TriplesMap to a logical table. The object of this predicate must be a resource that is further described using `rr:tableName` or `rr:sqlQuery`. A TriplesMap must have exactly one logical table. In SML, the *FROM* clause serves the same purpose. Table 2 compares how to state a logical table in R2RML and SML.

rr:tableName ”person”	. . . From person
rr:tableName ”SCOTT.DEPT”	. . . From ”SCOTT.DEPT”
rr:sqlQuery ”””SELECT . . . ”””	. . . From ”””SELECT . . . ”””

Table 2: Comparison of attributes of rr:logicalTable with SML's FROM clause.

Creating RDF terms from logical tables.

Both SML and R2RML allow to express how to create RDF terms from the rows of the underlying logical table. In SML, the term constructor expressions of the WITH clause serve this purpose, whereas R2RML introduces the notion of *TermMaps.* SML uses an expression syntax to specify the RDF term creation, which corresponds to using a combination of the properties `rr:template`, `rr:termType` and `rr:datatype` in R2RML. The template syntax for values of `rr:template` corresponds to the SML expression symbols *concat* and *urlEncode.* Table 3 shows examples of RDF term construction in both mapping languages. The function *asTemplate(expr)* is assumed to yield the R2RML template for a given SML expression. Note that SML is slightly more expressive in this regard, as it allows e.g. nested urlEncodings.

Forming Quads from RDF Terms.

Once there exists a specification of how to create RDF terms from the rows of a logical table, these RDF terms need to be grouped to form quads. In SML, this is done using the CONSTRUCT clause, which re-uses the quads production rule of the SPARQL 1.1 standard. Hence, anyone familiar

SML RDF term constructor	R2RML term map

`bNode(?COL)`	`... [ rr:column "COL" ;`
	`rr:termType rr:blankNode ]`

`bNode(`*expr*`)`	`... [ rr:template "`*asTemplate(expr)*`" ;`
	`rr:termType rr:blankNode ]`

`uri(`*expr*`)`	`... [ rr:`*(constant|column|template)* `"`*asTemplate(expr)*`";`
	`rr:termType rr:IRI ]`

`plainLiteral(?COL)`	`... [ rr:column "COL" ]`
`plainLiteral(`*expr*`)`	`... [ rr:template "`*asTemplate(expr)*`" ]`

`typedLiteral(?COL,` *xsd:int*`)`	`... [ rr:column "COL" ;`
	`rr:datatype` *xsd:int* `]`

`typedLiteral(`*expression*`,` *xsd:int*`)`	`... [ rr:template "`*asTemplate(expr)*`" ;`
	`rr:datatype` *xsd:int* `]`

Table 3: Transformation of SML term constructors to R2RML term maps

[18] http://www.w3.org/TR/sparql11-query/#rQuads

SML

```
Prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix ex: <http://ex.org/>

Create View hotels As
  Construct {
    ?s a ex:Hotel ;
      rdfs:label ?l ;
      ex:vacancy ?v
  }
  With
    ?s = uri(?website)
    ?l = plainLiteral(?name,’en’)
    ?v = typedLiteral(?vacancy,
             xsd:boolean)
  Constrain
    ?s prefix "http://ex.org/"
  From
    """SELECT website, name,
      vacancy FROM hotels"""
```

R2RML

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://ex.org/> .

<HotelTriplesMap>
  rr:logicalTable [ rr:sqlQuery
  """SELECT website, name,
  vacancy  FROM hotels""" ];

  rr:subjectMap [
    rr:column "website";
    rr:class ex:Hotel
  ];
  rr:predicateObjectMap [
    rr:predicate rdfs:label;
    rr:objectMap
      [ rr:column "name";
        rr:language "en"];
  ];
  rr:predicateObjectMap [
    rr:predicate ex:vacancy;
    rr:objectMap
      [ rr:column "vacancy";
        rr:datatype
          xsd:boolean ];
  ].
```

Figure 1: A simple view in SML and R2RML.

with SPARQL should already be familiar with SML in this regard.

In R2RML, the TriplesMap serves this purpose, however its specification is more verbose: For relating a TriplesMap to TermMaps for the subject, predicate, object and graph components, there exist the general properties `rr:subjectMap`, `rr:predicateMap`, `rr:objectMap` and `rr:graphMap`, respectively. Note that for each of these properties there exists a syntactic shortcut without the *Map* in the name, that can be used for constants.

These properties are used to form the following structure: Every TriplesMap carries exactly one specification for its subjects, and zero or more specifications for the pairs or predicates and objects. Subjects are specified by relating the TriplesMap to a TermMap using `rr:subjectMap`. A SubjectMap may carry zero or more attributes of *rr:graphMap* and *rr:class*. The former specifies in which graphs the generated triples reside. The latter is a syntactic shortcut for *rdf:type*'ing the subjects with given IRIs. Zero or more *rr:predicateObjectMap* attributes denote the predicate-object-pairs to associate with each subject. Thereby, each PredicateObjectMap carries the attributes *rr:objectMap* and *graphMap*, which again are TermMaps.

Foreign Key Relations among Logical Tables.

R2RML offers a model for the expression of joins. This model is primarily intended for the generation of IRIs that require a join of tables between one or more foreign key constraints to hold. The R2RML vocabulary distinguishes the roles of the *parent* and *child* table, where the child references the parent on a certain *join condition.*

SML offers a limited SQL syntax for this purpose, which is shown and compared to R2RML in Figure 2. Note, that in contrast to using SML's triple double-quotes syntax for stating SQL queries, this syntax allows the SML processor to understand the SQL natively (i.e. without requiring an full SQL parser) and thus consider it for optimizations such as self join elimination.

Assigning Triples to Named Graphs.

To assign triples to be generated to a certain named graph, again term maps are utilized. Accordingly, an additional term map inside a subject or predicate map is defined, which is called *graph map.* These nested term map expressions introduce further complexity to the actual triples map.

6. CONVERTING SML TO R2RML

In this section, we briefly outline the approach for the conversion of SML to R2RML. The process of converting prefix definitions is straightforward since the only difference is that SML uses the `Prefix` keyword, whereas in R2RML turtle notation `@prefix` is used. The name of an SML view definition serves as a base for crafting the IRIs for naming triples maps. However, it has to be taken into account that one SML view definition can correspond to many R2RML triples maps. The definition of a logical tables as table names, view names or queries have direct counterparts in R2RML, namely `rr:tableName` and `rr:sqlQuery`.

The SML `CONSTRAINT` clause has no equivalent in R2RML and is thus omitted in the conversion. Note that constraints only act as hints that may be considered for improving performance.

The *Construct* and *With* sections of an SML do not directly translate to R2RML. In general, a new triples map can be created for each quad of the *Construct* section. However, if an SML view defines multiple quads with the same variable in the subject position, multiple instances of *rr:predicateObjectMap* can be created on the same triples map.

Regarding the constructor arguments, one has to differentiate between an atomic value and a compound expression. An atomic value can either be a column reference, resulting in an *rr:column* term map or a constant expression requiring *rr:constant.* In all other cases, a more complex expression is assumed, resulting in an *rr:template.* Such an expression has to be evaluated from the innermost to the outermost term constructor which leads to the evaluated expression shown in Table 3.

SML

```
Prefix ex: <http://ex.org/>

Create View departments As
  Construct {
    ?d a ex:Department
  }
  With
    ?d = uri(ex:dept, ?name)
  From
    departments

Create View employees As
  Construct {
    ?e a ex:Employee ;
      ex:worksIn ?d
  }
  With
    ?e = uri(ex:emp, ?e.id)
    ?d = uri(ex:dept, ?d.name)
  From
    employees e Join departments d
      On (d.id = e.dept_id)
```

R2RML

```
@prefix ex: <http://ex.org/> .

<#Departments>
  rr:logicalTable [ rr:tableName "departments" ];

  rr:subjectMap [
      rr:template "http://ex.org/dept/{id}" ];
      rr:class ex:Department
  .

<#Employees>
  rr:logicalTable [ rr:tableName "employees" ];

  rr:subjectMap [
      rr:template "http://ex.org/emp/{id}" ;
      rr:class ex:Employee ];

  rr:predicateObjectMap [
    rr:predicate ex:worksIn;
    rr:objectMap [
      rr:parentTriplesMap <#Departments>;
      rr:joinCondition [
        rr:child "dept_id";
        rr:parent "id";
      ];
    ];
  ].
```

Figure 2: A view in SML and R2RML with a referencing object map.

An even more complex R2RML mapping has to be created if an SML variable already used in subject position is also referred to in the object position of another `Construct` statement. In this case a referencing object map has to be used in R2RML. An example and the corresponding triples maps are shown in Figure 2. There, the definition of the `rr:joinCondition` can be omitted since both triples maps refer to the same logical table. Note, that R2RML only provides a language expression to declare referencing objects and not for e.g. predicates and graphs. A conversion from R2RML to SML proceeds in a similar fashion as described in this section, however details are omitted for brevity.

7. EVALUATION

We evaluated the SML mapping language to clarify the following questions:

1. Is SML easier to read than R2RML and does SML have a lower entry barrier than R2RML?
2. Can people understand SML mappings or R2RML mappings faster?
3. If given the choice, would people prefer SML or R2RML?

7.1 Experimental Setup

We set up a survey, which consists of three parts:

1. Questions about prior expertise of the participant.
2. Test questions for SML and R2RML.
3. An assessment of the characteristics of SML and R2RML by the participants.

We used a standard star rating for most questions ranging from 1 star (lowest value) to 5 starts (highest value). The comparison with R2RML was performed as it is the current W3C standard for RDB2RDF mapping.

In the fist part, we asked participants to state their familiarity with the SML and R2RML languages as well as with related concepts such as the Turtle syntax and the SQL and SPARQL query languages. In the second part, we had 5 different tasks for participants. Each task was formulated for SML and R2RML with renamed classes and properties. In the first 3 tasks, participants had to select the subset of 4 shown triples, which was actually generated from a given mapping. The tasks were ordered by complexity of the mapping specification. In the 4th and 5th task, the inverse needed to be performed: Given a target RDF output, participants had to select those mappings from 4 presented mappings, which generates the target output. Finally, in the third part of the survey, participants had to assess a) the difficulty of the presented tasks, b) whether they could make sense of the SML and R2RML mappings, c) whether they found SML and R2RML easy to read, d) whether they would consider using SML for RDB2RDF tasks and e) whether they have a preference between SML and R2RML.

The survey was distributed to the Semantic Web and Linked Data mailing lists and announced on Twitter.

7.2 Results

Overall, a total of 102 participants took part in the survey, of which 73 completed the survey. We removed entries with an completion time below 500 seconds in order to remove bot entries and carefully assessed the removed entries. 46 participants remained out of which 28 answered all test questions correctly. The 46 participants required an average time of 1243.1 seconds to complete the survey. As a result, the overall time valid participants spent on the survey was 953 minutes. All results of the survey can also be directly obtained and analysed at the SML project website.

The averaged results of the survey are shown in Table 4. The self assessment scores in Table 4 illustrate that the audience is interested in RDB2RDF conversions and that the participants are familiar with Turtle (TTL), SPARQL and SQL. R2RML familiarity is considerably lower with an average of 3 and SML relatively unknown (1.74).

We discuss each of our evaluation questions in turn:

Readability and Entry barrier: Figure 3 shows that SML appears to have a lower entry barrier than R2RML. Participants who were familiar with R2RML already judged both languages to have similar readability. However, par-

criterion	value	criterion	value
Relevance :	4.15	Average Understandability R2RML :	3.85
Familiarity TTL :	4.11	Average Understandability SML :	3.88
Familiarity SPARQL:	4.30	Average Readability R2RML :	3.26
Familiarity SQL :	4.33	Average Readability SML :	3.72
Familiarity R2RML :	3	Average Considering SML :	3.59
Familiarity SML :	1.74	Average Preference R2RML - SML:	3.26

Table 4: Averaged results of survey questions (1 = lowest rating, 5 = highest rating).

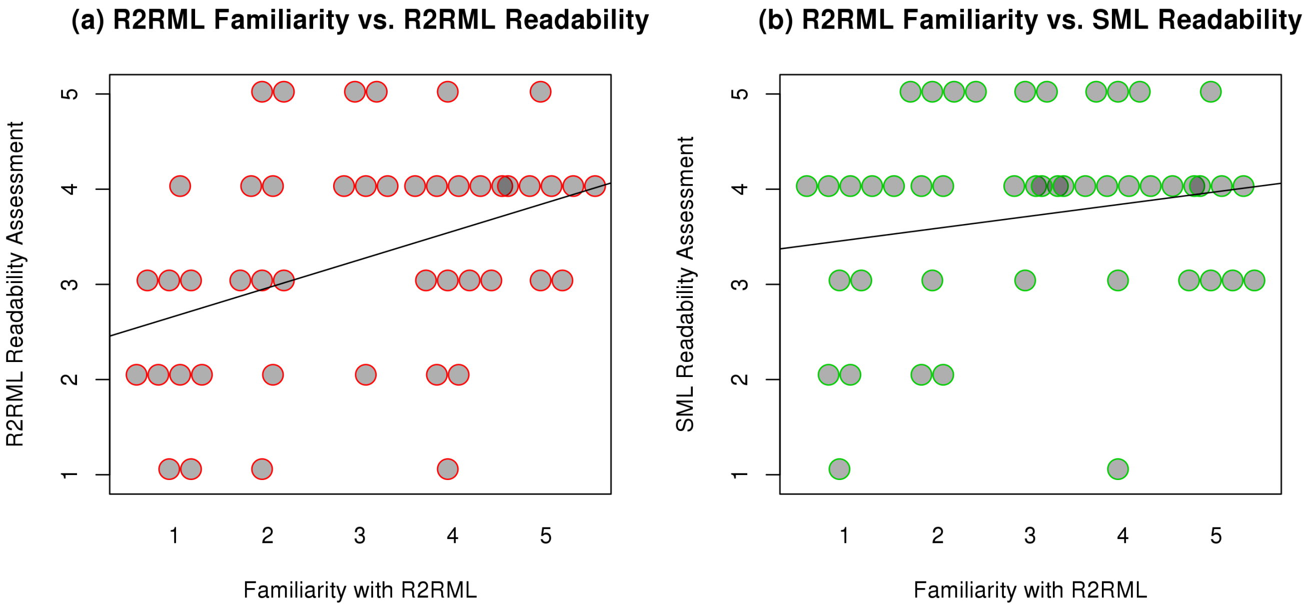

Figure 3: The plot shows the R2RML familiarity plotted against the readability assessment of R2RML and SML. Overall, SML was judged to be significantly more readable although this effect is reduced for participants already familiar with R2RML.

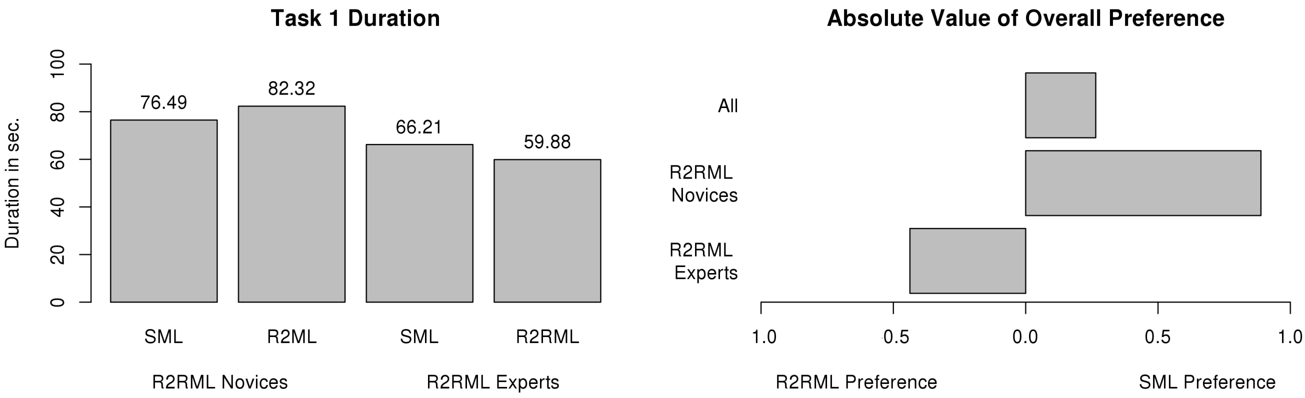

Figure 4: The figure on the left assesses the time needed to solve an RDB2RDF mapping task. The figure on the right shows the preference for a mapping language. Overall, SML tasks required less time and the language was preferred, although this does not hold when considering only R2RML experts.

ticipants who were less familiar with R2RML judged SML to be much more readable than R2RML and gave high readability scores.

Time needed to solve tasks: For this task, we randomly started the survey either with either an SML or R2RML task to be able to assess this evaluation question. The median time required for completing completing in R2RML was 67.08 seconds and for SML 69.23 seconds, giving R2RML a slight edge. In a more thorough analysis, Figure 4 shows the time required to solve the first RDB2RDF mapping task in the survey, for R2RML experts (self assessment greater or equal 4) and R2RML novices (self assessment less 4). R2RML experts require less time for solving the task in the R2RML, however R2RML novices appear are able to complete the task faster using SML. It is further clear that R2RML experts require less time for solving the task regardless of the utilized language. It should be noted that there is an unknown amount of time required to understand the task irrespective of the mapping language, i.e. the difference between the mapping language is larger than depicted.

Overall preference: Figure 4 shows that there is an overall preference for SML. This preference does not hold when considering only the group of R2RML experts, but is very significant when considering people not familiar with any of the two mapping languages.

8. CONCLUSIONS AND FUTURE WORK

In this article, we presented work towards a unified model for RDB2RDF mappings and the lightweight mapping language *SML* that reuses familiar elements of SPARQL and SQL in order to lower the learning curve and ease the manual writing and maintenance of view definitions. An extensive public survey confirmed that this is the case. We provided an in-depth comparison of how SML relates to the R2RML standard, and detailed how the former can be automatically converted to the latter.

SML has been successfully deployed in several scenarios: We created SML mappings for the BSBM and SP2 benchmarks, two popular SPARQL benchmarks that are often used for evaluating RDB2RDF mappers. Most prominently, we created SML mappings for transforming the OpenStreetMap (OSM) database to RDF. These efforts are carried out as part of the LinkedGeoData[19] (LGD) project, where we give access to more than 25 billion OSM RDF triples created through SPARQL-to-SQL rewriting over about 3 billion relational rows via more than 40 SML view definitions.

Furthermore, SML mappings have been created for two large scale linguistic resources: One is the mapping of the Wortschatz database[20], which contains statistics, such as frequency and co-occurrences, about words in more than 240 languages. The other resource is PanLex[21], which is a database holding translations of about 19 million expression extracted from over 2.000 sources. Links to the corresponding SML mappings are published together with our other SML related resources[22].

In general, we believe that mapping relational structures to RDF will stay a highly important topic in research and practice to provide an unobtrusive transition towards the use of semantic technologies. Providing engineers an intuitive yet powerful language is a crucial step to ease this transition. Future work will continue on extending the formalizations as well as sorting out details based on community feedb, such as whether an explicit `FROM QUERY` syntax for specifying SQL queries is preferred over the current approach where this is implied by the use of triple quotes.

Acknowledgment

This work was supported by grants from the EU's 7th Framework Programme provided for the projects LOD2 (GA no. 257943) and GeoKnow (GA no. 318159).

[19] http://linkedgeodata.org/

[20] http://www.wortschatz.uni-leipzig.de/

[21] http://ld.panlex.org

[22] http://sml.aksw.org

9. REFERENCES

[1] Mapping relational data to rdf with virtuoso's rdf views. http://virtuoso.openlinksw.com/whitepapers/relational%20rdf%20views%20mapping.html.

[2] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda. R2rml: Rdb to rdf mapping language (w3c recommendation). Technical report, 2012.

[3] C. Bizer and R. Cyganiak. D2r server – publishing relational databases on the semantic web. Poster at the 5th Int. Semantic Web Conf. (ISWC2006), 2006.

[4] S. Das, S. Sundara, and R. Cyganiak. R2rml: Rdb to rdf mapping language (w3c recommendation). Technical report, 2012.

[5] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. Rml: a generic language for integrated rdf mappings of heterogeneous data. In *Proceedings of the 7th Workshop on Linked Data on the Web (LDOW2014), Seoul, Korea*, 2014.

[6] T. B. Lee. Relational databases on the semantic web, 09 1998. Design Issues (published on the Web).

[7] C. Pinkel, C. Binnig, P. Haase, C. Martin, K. Sengupta, and J. Trame. How to best find a partner? an evaluation of editing approaches to construct r2rml mapping. In *ESWC*, 2014.

[8] F. Priyatna, O. Corcho, and J. Sequeda. Formalisation and experiences of r2rml-based sparql to sql query translation using morph. In *Proceedings of the 23rd international conference on World wide web*, pages 479–490. International World Wide Web Conferences Steering Committee, 2014.

[9] M. Rodriguez-Muro, M. Rezk, J. Hardi, M. Slusnys, T. Bagosi, and D. Calvanese. Evaluating sparql-to-sql translation in ontop. In *OWL Reasoner Evaluation Workshop*, volume 1015 of *CEUR Workshop Proceedings*, pages 94–100. CEUR-WS.org, 2013.

[10] M. Rodriguez-Muro, M. Rezk, J. Hardi, M. Slusnys, T. Bagosi, and D. Calvanese. Evaluating sparql-to-sql translation in ontop. 2013.

[11] K. Sengupta, P. Haase, M. Schmidt, and P. Hitzler. Editing r2rml mappings made easy. 2013.

[12] K. Sengupta, P. Haase, M. Schmidt, and P. Hitzler. Editing r2rml mappings made easy. In *International Semantic Web Conference (Posters & Demos)*, volume 1035 of *CEUR Workshop Proceedings*, pages 101–104. CEUR-WS.org, 2013.

[13] J. F. Sequeda and D. P. Miranker. Ultrawrap: Sparql execution on relational data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 22:19–39, 2013.

[14] J. Unbehauen, S. Hellmann, S. Auer, and C. Stadler. Knowledge extraction from structured sources. In S. Ceri and M. Brambilla, editors, *Search Computing - Broadening Web Search*, volume 7538 of *Lecture Notes in Computer Science*, pages 34–52. Springer, 2012.

[15] J. Unbehauen, C. Stadler, and S. Auer. Accessing relational data on the web with sparqlmap. In *JIST*, 2012.