Open access • Proceedings Article • DOI:10.1145/2063518.2063522

A comparison of RDB-to-RDF mapping languages — Source link

Matthias Hert, Gerald Reif, Harald C. Gall

Institutions: University of Zurich

Published on: 07 Sep 2011 - International Conference on Semantic Systems

Topics: Web mapping, Data mapping, SPARQL, RDF and Semantic Web Stack

A Comparison of RDB-to-RDF Mapping Languages

Hert, Matthias ; Reif, Gerald ; Gall, Harald C

Abstract: Mapping Relational Databases (RDB) to RDF is an active field of research. The majority of data on the current Web is stored in RDBs. Therefore, bridging the conceptual gap between the relational model and RDF is needed to make the data available on the Semantic Web. In addition, recent research has shown that Semantic Web technologies are useful beyond the Web, especially if data from different sources has to be exchanged or integrated. Many mapping languages and approaches were explored leading to the ongoing standardization effort of the World Wide Web Consortium (W3C) carried out in the RDB2RDF Working Group (WG). The goal and contribution of this paper is to provide a feature-based comparison of the state-of-the-art RDB-to-RDF mapping languages. It should act as a guide in selecting a RDB-to-RDF mapping language for a given application scenario and its requirements w.r.t. mapping features. Our comparison framework is based on use cases and requirements for mapping RDBs to RDF as identified by the RDB2RDF WG. We apply this comparison framework to the state-of-the-art RDB-to-RDF mapping languages and report the findings in this paper. As a result, our classification proposes four categories of mapping languages: direct mapping, read-only general-purpose mapping, read-write general-purpose mapping, and special-purpose mapping. We further provide recommendations for selecting a mapping language.

DOI: https://doi.org/10.1145/2063518.2063522

A Comparison of RDB-to-RDF Mapping Languages

Matthias Hert
Department of Informatics
University of Zurich
hert@ifi.uzh.ch

Gerald Reif
innovation process technology
gerald.reif@ipt.ch

Harald C. Gall
Department of Informatics
University of Zurich
gall@ifi.uzh.ch

ABSTRACT

Mapping Relational Databases (RDB) to RDF is an active field of research. The majority of data on the current Web is stored in RDBs. Therefore, bridging the conceptual gap between the relational model and RDF is needed to make the data available on the Semantic Web. In addition, recent research has shown that Semantic Web technologies are useful beyond the Web, especially if data from different sources has to be exchanged or integrated. Many mapping languages and approaches were explored leading to the ongoing standardization effort of the World Wide Web Consortium (W3C) carried out in the RDB2RDF Working Group (WG). The goal and contribution of this paper is to provide a feature-based comparison of the state-of-the-art RDB-to-RDF mapping languages. It should act as a guide in selecting a RDB-to-RDF mapping language for a given application scenario and its requirements w.r.t. mapping features. Our comparison framework is based on use cases and requirements for mapping RDBs to RDF as identified by the RDB2RDF WG. We apply this comparison framework to the state-of-the-art RDB-to-RDF mapping languages and report the findings in this paper. As a result, our classification proposes four categories of mapping languages: direct mapping, read-only general-purpose mapping, read-write general-purpose mapping, and special-purpose mapping. We further provide recommendations for selecting a mapping language.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.1 [**Models and Principles**]: Miscellaneous

General Terms

Documentation, Languages

Keywords

RDB-to-RDF mapping, feature-based comparison

I-SEMANTICS 2011, 7th Int. Conf. on Semantic Systems, Sept. 7-9, 2011, Graz, Austria

1. INTRODUCTION

Mapping Relational Databases (RDB) to RDF is an active field of research. As reported in [14], the majority of data on the current Web is stored in RDBs. Therefore, bridging the conceptual gap between the relational model and RDF is needed to make the data available on the Semantic Web. In addition, recent research has shown that Semantic Web technologies are useful beyond the Web, especially if data from different sources has to be exchanged or integrated (*e.g.,* [31, 28, 27]).

Many approaches were explored to make relational data available to Semantic Web-enabled applications. Depending on the requirements, these approaches introduced mapping languages that range from simple and pragmatic to highly specific or general-purpose. This lead to the ongoing standardization effort of the World Wide Web Consortium (W3C) carried out in the RDB2RDF Working Group[1] (WG). The mission of the RDB2RDF WG as defined in their charter [20] is *to standardize a language for mapping relational data and relational database schemas into RDF and OWL*. Although a standard mapping language is under development, we argue in this paper that alternative languages still have a right to exist as they may provide features or simplicity required in certain use cases that the standard mapping language cannot provide or explicitly excludes.

The goal and contribution of this paper is to provide a feature-based comparison of the state-of-the-art RDB-to-RDF mapping languages. It should act as a guide in selecting a RDB-to-RDF mapping language for a given application scenario and its requirements w.r.t. mapping features. Our comparison framework is based on use cases and requirements for mapping RDBs to RDF as identified by the RDB2RDF WG. We apply this comparison framework to the state-of-the-art RDB-to-RDF mapping languages and report the findings in this paper. As a result, our classification proposes four categories of mapping languages: direct mapping, read-only general-purpose mapping, read-write general-purpose mapping, and special-purpose mapping. We further provide recommendations for selecting a mapping language.

The remainder of this paper is structured as follows. Section 2 presents existing surveys related to RDB-to-RDF mapping and Section 3 introduces the mapping languages covered in this comparison. The framework we use for comparing the mapping languages is defined in Section 4. Features are extracted from the document *Use Cases and Requirements for Mapping Relational Databases to RDF* [32] produced by the RDB2RDF WG. Additional features which

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

recently gained attention in the research community [23, 7, 11] complement the comparison framework. In Section 5, we apply the comparison framework to the presented mapping languages and we discuss the results on a feature-by-feature basis. Section 6 classifies the mapping languages into four categories and provides recommendations for selecting a mapping language.

2. RELATED WORK

The W3C RDB2RDF Incubator Group[2] (XG) produced *A Survey of Current Approaches for Mapping of Relational Databases to RDF* [34] as part of their mission. It surveys current techniques, tools, and applications for mapping between RDBs and RDF. A survey reference framework is defined that covers aspects such as mapping creation, representation and accessibility, application domain, support for data integration as well as the implementation of the mapping, *i.e.,* static Extract Transform Load (ETL) versus dynamic query translation. Compared to this paper the survey of the RDB2RDF XG has a different scope. While we focus on the individual *features* of the mapping languages, the RDB2RDF XG survey is focused on the overall approach and implementation of the mappings. The mapping *languages* are only briefly addressed on a higher level and are not compared on a feature-by-feature basis.

The W3C Semantic Web Advanced Development for Europe[3] (SWAD-Europe) dedicates two brief sections of their Deliverable 10.2 [5] to RDB-to-RDF mapping. It contains general remarks on RDB-to-RDF mapping and refers to two implemented mapping tools. Due to the early publication of this report in 2003, it could not include many of the mapping languages covered in this paper because they were developed and/or published after the release of the SWAD-Europe deliverable.

3. MAPPING LANGUAGES

In this section, we briefly introduce the mapping languages covered by this comparison. To be included, a mapping language needs to *a)* have a clear focus on mapping RDBs (*i.e.,* other tabular data such as spreadsheets are not covered) and *b)* be general applicable (*i.e.,* not a domain-specific solution).

Direct Mapping: In [6], a direct approach for mapping RDBs to the Semantic Web is proposed. It maps relational tables to classes in an RDF vocabulary and the attributes of the tables to properties in the vocabulary. The goal is to expose a RDB on the (Semantic) Web to make extra statements about it. The URIs of the instances as well as those of the vocabulary classes are generated automatically based on the RDB schema and data.

The focus of [25] is on the automatic discovery of mappings and not on their representation. The result is a simple table-to-class and attribute-to-property mapping extended with heuristics to find implicit subclass relationships in the RDB schema.

SquirrelRDF [37] is another implementation of the direct mapping as proposed in [6]. Its mapping is raw, *i.e.,* the classes and properties of the target RDF vocabulary are generated from the names in the RDB schema. Mapping to a domain ontology is postponed to a later stage and is performed by RDF-based tools.

All three mappings use a similar direct approach for RDB-to-RDF mapping. We therefore summarize them under the term *Direct Mapping*.

eD2R: The case study described in [3] uses eD2R for the RDB-to-RDF mapping. eD2R is an extension of D2R MAP [9] with the goal of covering mapping situations involving databases that are lightly structured or not in first normal form [18]. The mappings are based on SQL queries that extract records from the RDB and transformation functions that can be applied to the extracted values. Existing vocabularies can be reused. eD2R extends the XML-based syntax of D2R MAP to represent the mappings.

R_2O: R_2O [4] is an extensible and fully declarative language to describe mappings between RDB schemata and ontologies implemented in RDFS or OWL. It is assumed that the RDB and ontology models are pre-existing. R_2O is aimed at situations where the similarity between the ontology and the RDB model is low. It has been conceived to be expressive enough to cope with complex mapping cases where one model is richer, more generic/specific, or better structured than the other. Mappings are expressed in a XML-based syntax.

Relational.OWL: In [16], a OWL-based representation format for relational data and schema components, called Relational.OWL, is introduced. It defines a OWL Full ontology to describe the schema and data of a RDB. The target application of this mapping is data exchange in peer-to-peer databases.

Virtuoso RDF Views: The Virtuoso Universal Server by Openlink Software features RDF Views [17, 36] to expose relational data on the Semantic Web. It consists of a declarative Meta Schema Language for defining the mapping of SQL data to preexisting RDF vocabularies. At the most basic level, Virtuoso RDF Views transform the result set of a SQL SELECT query into a set of triples. The Meta Schema Language resembles SQL DDL from a syntax point of view.

D2RQ: D2RQ [12, 10] is a mapping language and platform for treating non-RDF relational databases as virtual RDF graphs. Its aim is to expose RDBs on the Semantic Web to provide access via SPARQL queries and Linked Data. Existing RDF vocabularies can be reused. The mappings are expressed in RDF and formally defined by an RDFS schema. It is the successor to the XML-based D2R MAP [9].

Triplify: Triplify [1] is a light-weight approach to publish Linked Data from RDBs. It is based on mapping HTTP-URI requests onto RDB queries and translating the resulting relations into RDF statements. The main motivation of Triplify is that the majority of information on the Web is already stored in structured form (*i.e.,* as data in RDBs) but published as HTML by Web applications (*e.g.,* CMS, Wiki, Blog). Mapping

[2] `http://www.w3.org/2005/Incubator/rdb2rdf/`
[3] `http://www.w3.org/2001/sw/Europe/`

the RDB schemata of such popular Web applications results in a boost of Semantic Web adoption as these Web applications are deployed many times. Triplify mappings are implemented as PHP scripts.

R2RML: R2RML [15] is the mapping language of the ongoing work by the W3C RDB2RDF WG to standardize RDB-to-RDF mappings. The goal is to define a vendor-independent mapping language for read-only data access.

R3M: R3M [24] is the mapping language of the OntoAccess[4] mediation platform [22]. As an update-aware mapping language it enables bidirectional RDF-based access to the RDB, *i.e.,* read and *write* access is supported. R3M employs a RDF-based syntax that contains the mappings of tables to classes and attributes to properties as well as information about integrity constraints.

4. COMPARISON FRAMEWORK

In this section, we introduce the framework used for comparing the RDB-to-RDF mapping languages presented above. It is based on the document *Use Cases and Requirements for Mapping Relational Databases to RDF* [32] of the W3C RDB2RDF Working Group[5] (WG). Extensions are made in the area of RDF-based write access, a feature not addressed by the W3C RDB2RDF WG that lately gained attention from the research community [23, 7, 11].

A direct approach for mapping RDBs to the Semantic Web was proposed in [6]. It maps (physical) relational tables to classes in an RDF vocabulary and relational attributes to properties in that vocabulary. We consider this to be the most basic mapping and we require that a mapping language supports at least this kind of mapping to be included in the comparison. Therefore, these two features are not explicitly represented in the comparison framework but are assumed to hold implicitly.

We now enumerate the features that define the framework our comparison of RDB-to-RDF mapping languages is based on.

F1 Logical Table to Class: A logical table is defined as a SQL view already stored in the RDB system or the result of an ad-hoc SQL query, both resulting in a table that is not necessarily stored physically in the RDB. Feature F1 enables the mapping of such a logical table to a class in the RDF vocabulary.

F2 M:N Relationships: RDBs require a special construct called link (or join) tables to represent M:N relationships among concepts. RDF, however, does not require such helper constructs. Therefore, link tables should be mapped to RDF properties instead of classes. Feature F2 enables the mapping of link tables to properties in the RDF vocabulary.

F3 Project Attributes: Tables in a RDB may contain attributes that should not be part of the RDF representation (*e.g.,* irrelevant or sensitive attributes such as passwords). A mapping should project only the required attributes to the RDF representation. Feature F3 enables projecting a subset of the attributes in the mapping.

F4 Select Conditions: RDB tables may contain records that should not be part of the RDF representation (*e.g.,* outdated data). A mapping should support the definition of a condition that is evaluated for each record to decide about its inclusion in the RDF representation. Feature F4 enables the definition of such select conditions in the mapping.

F5 User-defined Instance URIs: Records in the RDB are converted to RDF instances identified by an URI. These instance URIs can be automatically generated based on the RDB schema and data or the user of the RDB-to-RDF mapping may be able to define the (syntactic) form of the generated URIs. Feature F5 enables user-defined instance URIs.

F6 Literal to URI: URIs as a datatype are typically not supported in a RDB system. Therefore, values representing URIs are stored as character literals (*e.g.,* email addresses). Such literal values should be converted to valid URIs in the RDF representation. Feature F6 enables the generation of URIs from literal values.

F7 Vocabulary Reuse: The vocabulary terms that a RDB schema is mapped to can be generated automatically (based on the names of tables and attributes in the RDB schema) or existing RDF vocabularies can be reused. Feature F7 enables the mapping to existing RDF vocabulary terms.

F8 Transformation Functions: Literal values may require a different (syntactic) representation in RDF (*e.g.,* temperature in Centigrade vs. Fahrenheit). Transformation functions may be defined to provide the conversion of values between the RDB and RDF representations. Feature F8 enables support for such transformation functions.

F9 Datatypes: Datatypes of literal values are an important feature in RDB systems and RDF. Although there exist mappings [26] of common SQL datatypes to the XML datatypes [8] used in RDF, the information about datatypes might be of value. Feature F9 enables the explicit representation of datatype information in the mapping.

F10 Named Graphs: RDF data sets may consists of multiple named graphs. A mapping may therefore assign certain parts of a RDB to a specific named graph. Feature F10 enables the support of named graphs in the RDB-to-RDF mapping.

F11 Blank Nodes: Blank nodes are used in RDF to represent instances that have no RDF URI reference identifier but are distinct in an RDF graph [29], *i.e.,* they are a form of existential quantification [21]. One common usage of blank nodes is in structured property values (*e.g.,* structuring an address consisting of street, postal code, and city). In the case of RDB-to-RDF mapping they may also be used to represent RDB records without a primary key. Feature F11 enables support for generating blank nodes.

[4] `http://ontoaccess.org/`

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

F12 Integrity Constraints: Integrity constraints provide a basic mechanism of semantics in RDBs. We distinguish between key constraints (primary key, foreign key) and other constraints (not null, unique, check). Feature F12 enables the explicit description of constraints in a mapping language.

F13 Static Metadata: Static metadata may be added to the RDF representation that has no direct counterpart in the RDB (*e.g.,* provenance or licensing information). Schema-level triples such as *rdf:type* triples and triples originating from the target RDF vocabulary are, however, not in the scope of this feature. Feature F13 enables the definition of static metadata.

F14 One Table to *n* Classes: Mapping a single table to multiple classes in the RDF vocabulary may be necessary if the RDB schema is *a)* not normalized or *b)* concept specialization is encoded as an attribute of the table. This results in two mapping cases: *a)* the table is mapped multiple times, each time with a subset of the attributes and *b)* records of the table are mapped to a different class depending on the value of a specific discriminator attribute. Feature F14 enables the mapping of one table to n classes.

F15 Write Support: RDF data, including mapped RDBs, are often accessed in a read-only manner (*e.g.,* via SPARQL queries or Linked Data). However, support for write access is required in certain use cases which lately gained attention from the research community [23, 7, 11]. The requirements of write access should be explicitly addressed in a mapping language. Feature F15 enables explicit support for RDF-based write access to relational data.

Based on this set of fifteen features we compare the mapping languages presented in Section 3 and discuss important differences in feature support and the resulting consequences for RDB-to-RDF mapping systems.

5. DISCUSSION

In this section, we discuss the RDB-to-RDF mapping languages presented in Section 3. The discussion is structured according to the features of the comparison framework introduced in Section 4. We mostly limit the discussion to the mapping language that either support a feature partially or not at all. If a mapping language is not mentioned in the discussion of a feature, the reader may assume that it supports the feature. See also Table 1 for a summary.

F1 Logical Table to Class: Mapping logical tables to ontology classes is supported in one form by all of the mapping languages. This is the case in the mapping of existing views as they can be treated like a physical table. The Direct Mapping does not support mapping the results of an ad hoc query to a class. Relational.OWL does not mention the mapping of logical tables due to its focus on representing the core database schema and data. R3M does not support mapping the results of a query. Mapping of existing views is not prohibited, although this may interfere with R3M's main motivation of bidirectional data access (*cf.* the view update problem [2]).

F2 M:N Relationships: R3M is the only mapping language to provide explicit support for M:N relationships. The Direct Mapping, R_2O, and Relational.OWL do not mention any special support for mapping link tables to ontology properties. The other mapping languages provide implicit support due to their use of SQL (or fragments thereof) as an essential part of the mapping language.

F3 Project Attributes: The Direct Mapping automatically maps every attribute to properties, projecting only a subset of attributes is not intended. Virtuoso, Triplify as well as R2RML delegate the projection of attributes to the SQL queries used for the mapping.

F4 Select Conditions: The Direct Mapping does not provide the selection of rows via a condition. Also Relational.OWL does not allow this. Virtuoso, Triplify, and R2RML again use SQL with its powerful support for conditions to implement this feature. R3M provides limited support for the feature as solely conditions with equality are allowed to preserve the support for bidirectional data access.

F5 User-defined Instance URIs: Instance URIs are generated automatically in the Direct Mapping, hence customization by the user is not supported. Likewise, Relational.OWL with its focus on data exchange does not intend user-defined instance URIs, in fact, blank nodes are used for all instances. Triplify relies on the string concatenation feature of SQL to define custom instance URIs.

F6 Literal to URI: No support for this feature is provided by the Direct Mapping as well as Relational.OWL. All other mapping languages support it either with explicit language constructs or in the case of Triplify and R2RML via the SQL-based mapping.

F7 Vocabulary Reuse: The Direct Mapping itself does not support the reuse of existing vocabulary terms. Properties are generated based on the attribute names in the database schema. Existing classes can only be used by adding corresponding *rdf:type* statements in an additional step. Relational.OWL does also not provide support for vocabulary reuse.

F8 Transformation Functions: Transformation functions are not in the scope of the Direct Mapping and Relational.OWL. Triplify and R2RML use functions in SQL to transform object values. R3M requires the functions to be defined bidirectionally to retain read and write data access.

F9 Datatypes: The Direct Mapping is the only mapping language that does not describe the explicit representation or the mapping of datatypes.

F10 Named Graphs: Many of the mapping languages (Direct Mapping, eD2R, R_2O, and D2RQ) predate the introduction of named graphs and therefore provide no support for this feature. Others (Relational.OWL, Triplify, and R3M) explicitly choose not to support named graphs for various reasons.

Table 1: Summary table of RDB-to-RDF mapping language comparison

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
Direct Mapping	(✔)	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✔
eD2R	✔	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	(✔)	✘	✔	✘
R_2O	✔	✘	✔	✔	✔	✔	✔	✔	✔	✘	✔	(✔)	✘	(✔)	✘
Relational.OWL	(✔)	✘	✔	✘	✘	✘	✘	✘	✔	✘	✔	(✔)	✘	✘	✔
Virtuoso	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	(✔)	✘	✔	✘
D2RQ	✔	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	(✔)	✔	✔	✘
Triplify	✔	✔	✔	✔	✔	✔	✔	✔	✔	✘	✘	(✔)	✘	✔	✘
R2RML	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	(✔)	✔	✔	✘
R3M	(✔)	✔	✔	(✔)	✔	✔	✔	✔	✔	✘	✘	✔	✘	✔	✔

✔ = full support (✔) = partial support ✘ = no support

F11 Blank Nodes: Instance URIs in the Direct Mapping are generated automatically for each database record and therefore blank nodes are not supported. In Relational.OWL, all instances are represented as blank nodes because individual URIs are not needed for its main application scenario as a data exchange format. Support for blank nodes is not described in Triplify and R3M.

F12 Integrity Constraints: No support for integrity constraints is available in the Direct Mapping. R3M provides rich support for key and the other constraints. All other mapping languages are limited to key constraints.

F13 Static Metadata: Generating triples on the schema level (*e.g., rdf:type* triples) is supported by many of the mapping languages but is not in the scope of this feature as explained in Section 4. Support for static, non-schema-level triples (*e.g.,* provenance or licensing information) is only available in D2RQ and R2RML.

F14 One Table to n Classes: The Direct Mapping as well as Relational.OWL do not allow the mapping of a single table to multiple classes. R_2O provides partial support as mapping a table multiple times with a subset of the attributes is possible while mapping a table multiple times based on the value of a discriminator attribute is not. Virtuoso, D2RQ, Triplify, and R2RML again rely on the power of a SQL-based mapping to support this feature. R3M allows the mapping of one table to n classes in a restricted form. To preserve bidirectional data access, the constraints defined on a table must not be violated in a mapping. For example, if a table contains an attribute with a not null constraint, this attribute must be mapped to each class because it must be set for each record. Otherwise, it is not possible to insert any instances of this class.

F15 Write Support: Support for RDF-based write access to the RDB is influenced by multiple of the other features, but some mapping languages address write support explicitly while others choose a feature set that renders write support impractical. The Direct Mapping could support write access as it represents the RDB schema directly in RDF. However, none of the existing approaches that apply the Direct Mapping consider write access explicitly in their feature set. Relational.OWL with its data exchange background obviously provides write support. R3M as a bidirectional approach does so as well. The other mapping languages were all designed with read-only use cases in mind. Attempts were made to add write support to some of the read-only languages (*e.g.,* D2RQ/Update[6] for D2RQ or [19] for R2RML), but it was also shown that this requires restrictions to the mapping languages resulting in existing mapping definitions to be no longer valid in the restricted approaches.

6. CONCLUSION

In this paper, we presented a feature-based comparison of the state-of-the-art RDB-to-RDF mapping languages.

Based on feature support, the mapping languages are classified into four categories: direct mapping, read-only general-purpose mapping, read-write general-purpose mapping, and special-purpose mapping.

Direct Mapping: The direct mapping, as its name implies, is a direct approach for RDB-to-RDF mapping. Aimed at providing simple means to map RDBs to RDF it does not fully support any of the additional features used in this comparison. Therefore, peculiarities of the relational model such as link tables remain in the RDF representation, although RDF does note require such helper constructs as it provides direct means to model these relationships. The advantage of this mapping is its simplicity to understand and implement the language. It is therefore recommended in application scenarios where a direct representation of the relational schema is acceptable and simplicity is of higher value.

Read-only General-purpose Mapping: Read-only general-purpose mapping languages (Virtuoso, D2RQ, and R2RML) are very similar w.r.t. the features they support. The differences are in features not directly related to the expressiveness of the mappings, namely support for named graphs (F10) and static metadata (F13). Virtuoso provides support for named graphs

[6] `http://d2rqupdate.cs.technion.ac.il/`

(F10) but not for static metadata (F13). The opposite is the case in D2RQ, no support for named graphs (F10) is provided, but static metadata (F13) can be specified. R2RML supports both features.

These mapping languages enable a highly expressive bridging of the conceptual gap between RDF and the relational model. However, this higher expressiveness also implies an increased complexity that results in an unidirectional mapping, *i.e.,* bidirectional read and write access to the data is impractical. Understanding and implementing the mapping languages may also require a higher learning effort. Due to their high expressiveness, mapping languages of this category can be recommended for various application scenarios as long as the usage is limited to read-only data access.

In choosing a specific mapping language of this category, aspects not related to feature support may influence the decision such as implementation maturity, standards compliance, or licensing terms.

Read-Write General-purpose Mapping: R3M can be classified as a general-purpose mapping language as well, but it has the additional goal of providing bidirectional (*i.e.,* read and write) data access to the RDB. This explains most of the differences in feature support compared to the read-only general-purpose mapping languages. The most important difference is that in R3M the mapping of logical tables (F1) can not be allowed arbitrarily due to the view update problem [2]. The definition of select conditions (F4) that decide about the inclusion of a record in the RDF representation must also be restricted to preserve write access. Conditions based on inequalities (*e.g.,* less than $<$) result in ambiguities if new data should be inserted. For example, imagine a mapping where a table *person* that represents people of all ages should be (partially) mapped to a concept *Adult* that represents people of the age 18 or older. For this, a select condition must be defined on the attribute *year_of_birth* of the *person* table, namely that its value is less than *1993*. If now a new instance of the concept *Adult* should be inserted into the RDB that does not explicitly contain a value for *year_of_birth*, it remains ambiguous what value should be set for this attribute as any value less than *1993* satisfies the select condition. Therefore, select conditions have to be restricted to equality ($=$) conditions. Furthermore, support for integrity constraints (F12) is extended to other constraints such as not null, unique, and check to enable the detection of invalid write requests, *e.g.,*, detecting missing data for a not null attribute.

In summary, the mapping language of this category provides a more expressive bridging of the conceptual gap between the relational model and RDF than the mapping approaches of the direct mapping. Peculiarities of the relational model such as link tables are not transferred to the RDF representation. This mapping language is, however, less expressive than the read-only general-purpose mapping languages, but this is required to guarantee support for write access. R3M as the sole mapping language of this category is recommended for application scenarios where RDF-based read and write access to the relational data is needed.

Special-purpose Mapping: Mapping languages such as eD2R, R_2O, and Triplify were developed for specific use cases and are obviously influenced by those use cases in the features they support. This does not necessarily result in a loss of expressiveness or applicability compared to the general-purpose languages. The differences are mostly limited to a few features such as support for named graphs (F10), blank nodes (F11), or static metadata (F13). None of the mapping languages in this category provide support for named graphs (F10) or static metadata (F13). Blank nodes (F11) are supported by eD2R and R_2O but not by Triplify. The use case of Relational.OWL, however, is highly specialized and does therefore neither implement nor require many of the described features.

The mapping languages of this category were developed for their specific application scenarios and are therefore recommended for application in closely related scenarios where the general-purpose mapping languages are not applicable or too complex.

Virtuoso, Triplify, R2RML, and to some extent D2RQ are mapping languages that rely heavily on SQL to implement the mapping. While on the one hand this yields certain advantages, it also entails serious drawbacks. The key benefit is that it allows one to reuse the power of the SQL language in defining views over the relational data. This pushes the main mapping work to the database system and therefore reduces implementation effort. On the other hand, there are two major drawbacks. First of all, in using SQL as the mapping language the semantics of the mapping is hidden in SQL strings and is therefore not easily accessible, *i.e.,* not without parsing the SQL strings. Second, mappings based on SQL views suffer from the same problem w.r.t. write access as standard SQL views, namely the view update problem [2]. History showed that trying to add write access to a mapping or a view definition language is in general impractical (*cf.* discussion of *F15 Write Support* in Section 5). Existing mapping or view definition languages would have to be restricted to a subset of the original language to provide general support for write access to the data (*e.g.,* [13]). This renders existing mapping/view definitions incompatible with the restricted languages. As a result, existing mapping/view definitions need to be rewritten, invalidating one of the top argument for reusing existing mappings. In certain mapping cases it might not even be possible to adapt the mapping because it uses some of the features that render the mapping language read-only.

The situation in the current read-only mapping languages resembles the introduction of SQL views where write access was also not addressed from the beginning. Even the ongoing standardization work by the W3C RDB2RDF WG explicitly defines write access to the data as out of scope [20]. This trend leads to a situation where RDB-to-RDF data sets are not on par with native RDF data sets, but limited to read-only application scenarios. Currently, there is no high demand for write access to RDF data as the present SPARQL 1.0 [33] recommendation is limited to read-only queries and no standard data manipulation approach for RDF exists. However, the upcoming SPARQL 1.1 recommendation includes the update language for RDF called SPARQL 1.1 Update [35] and the SPARQL 1.1 Graph Store HTTP Protocol [30], which will increase the demand for

write access. Write access should be possible irrespective of the source of the data being a native RDF triple store or a mediated RDB. Approaches such as R3M exist that address this problem.

In summary, we showed that based on feature support the state-of-the-art RDB-to-RDF mapping languages can be classified into four categories: direct mappings that provide simple means to represent RDB schemata and data in RDF; general-purpose mapping languages that provide highly expressive RDB-to-RDF mappings, but are limited to read-only data access; general-purpose mapping languages that are less expressive but enable a bidirectional (*i.e.,* read and write) data access; and special-purpose mapping languages with a feature set tailored to specific application scenarios. We further provided recommendations for selecting a mapping language.

7. ACKNOWLEDGMENTS

Partial support provided by Swiss National Science Foundation award number PDAMP2-122957.

8. REFERENCES

[1] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify – Light-Weight Linked Data Publication from Relational Databases. In *Proceedings of the 18th International World Wide Web Conference*, 2009.

[2] F. Bancilhon and N. Spyratos. Update Semantics of Relational Views. In *ACM Transactions on Database Systems*, 1981.

[3] J. Barrasa, O. Corcho, and A. Gómez-Pérez. Fund Finder: A Case Study of Database-to-Ontology Mapping. In *Proceedings of the Semantic Integration Workshop*, 2003.

[4] J. Barrasa, O. Corcho, and A. Gómez-Pérez. R2O, an Extensible and Semantically Based Database-to-Ontology Mapping Language. In *Proceedings of the 2nd Workshop on Semantic Web and Databases*, 2004.

[5] D. Beckett and J. Grant. SWAD-Europe Deliverable 10.2: Mapping Semantic Web Data with RDBMSes. `http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/`, January 2003. Last visited July 2011.

[6] T. Berners-Lee. Relational Databases on the Semantic Web. `http://www.w3.org/DesignIssues/RDB-RDF.html`, 2009. Last visited July 2011.

[7] T. Berners-Lee, R. Cyganiak, M. Hausenblas, J. Presbrey, O. Seneviratne, and O.-E. Ureche. Realising a Read-Write Web of Data. `http://web.mit.edu/presbrey/Public/rw-wod.pdf`, June 2009. Last visited July 2011.

[8] P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation. `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/`, October 2004.

[9] C. Bizer. D2R MAP – A Database to RDF Mapping Language. In *Proceedings of the 12th International World Wide Web Conference*, 2003.

[10] C. Bizer, R. Cyganiak, J. Garbers, O. Maresch, and C. Becker. The D2RQ Platform v0.7 - Treating Non-RDF Relational Databases as Virtual RDF Graphs - User Manual and Language Specification. `http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/20090810/`, August 2009.

[11] C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas. Linked Data on the Web - Topics of Interest. `http://events.linkeddata.org/ldow2011/`, 2011. Last visited July 2011.

[12] C. Bizer and A. Seaborne. D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs. In *Proceedings of the 3rd International Semantic Web Conference*, 2004.

[13] A. Bohannon, B. C. Pierce, and J. A. Vaughan. Relational Lenses: A Language for Updatable Views. In *Proceedings of the 25th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 2006.

[14] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured Databases on the Web: Observations and Implications. *SIGMOD Record*, 2004.

[15] S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF Mapping Language. W3C Working Draft. `http://www.w3.org/TR/2010/WD-r2rml-20101028/`, October 2010.

[16] C. P. de Laborda and S. Conrad. Relational.OWL – A Data and Schema Representation Format Based on OWL. In *Proceedings of the 2nd Asia-Pacific Conference on Conceptual Modelling*, 2005.

[17] O. Erling and I. Mikhailov. RDF Support in the Virtuoso DBMS. In *Proceedings of the SABRE Conference on Social Semantic Web*, 2007.

[18] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall Press, 2008.

[19] A. Garrote and M. N. M. Garcia. RESTful Writable APIs for the Web of Linked Data Using Relational Storage Solutions. In *Proceedings of the WWW2011 Workshop on Linked Data on the Web*, 2011.

[20] H. Halpin and I. Herman. RDB2RDF Working Group Charter. `http://www.w3.org/2009/08/rdb2rdf-charter`. Last visited July 2011.

[21] P. Hayes. RDF Semantics. W3C Recommendation. `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`, February 2004.

[22] M. Hert. Relational Databases as Semantic Web Endpoints. In *Proceedings of the 6th European Semantic Web Conference*, 2009.

[23] M. Hert, G. Reif, and H. C. Gall. 'Semantic Web 2.0' - Write-enabling the Web of Data. In *Proceedings of the 6th Workshop on Semantic Web Applications and Perspectives*, 2010.

[24] M. Hert, G. Reif, and H. C. Gall. Updating Relational Data via SPARQL/Update. In *EDBT Workshop Proceedings*, 2010.

[25] W. Hu and Y. Qu. Discovering Simple Mappings Between Relational Database Schemas and Ontologies. In *Proceedings of the 6th International and 2nd Asian Semantic Web Conference*, 2007.

[26] ISO/IEC. ISO/IEC 9075 Part 14: XML-Related Specifications (SQL/XML).

[27] A. Langegger, W. Wöss, and M. Blöchl. A Semantic Web Middleware for Virtual Data Integration on the Web. In *Proceedings of the 5th European Semantic Web Conference*, 2008.

[28] L. Ma, X. Sun, F. Cao, C. Wang, and X. Wang. Semantic Enhancement for Enterprise Data Management. In *Proceedings of the 8th International Semantic Web Conference*, 2009.

[29] F. Manola and E. Miller. RDF Primer. W3C Recommendation. `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`, February 2004.

[30] C. Ogbuji. SPARQL 1.1 Graph Store HTTP Protocol. `http://www.w3.org/TR/2011/WD-sparql11-http-rdf-update-20110512/`, May 2011.

[31] C. Patel, S. Khan, and K. Gomadam. TrialX: Using Semantic Technologies to Match Patients to Relevant Clinical Trials Based on Their Personal Health Records. In *Proceedings of the 8th International Semantic Web Conference*, 2009.

[32] E. Prud'hommeaux and M. Hausenblas. Use Cases and Requirements for Mapping Relational Databases to RDF. W3C Working Draft. `http://www.w3.org/TR/2010/WD-rdb2rdf-ucr-20100608/`, June 2010.

[33] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation. `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`, January 2008.

[34] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. T. Jr, S. Auer, J. Sequeda, and A. Ezzat. A Survey of Current Approaches for Mapping of Relational Databases to RDF. `http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf`, 2009. Last visited July 2011.

[35] S. Schenk, P. Gearon, and A. Passant. SPARQL 1.1 Update. W3C Working Draft. `http://www.w3.org/TR/2010/WD-sparql11-update-20101014/`, October 2010.

[36] O. Software. Mapping Relational Data to RDF with Virtuoso's RDF Views. `http://virtuoso.openlinksw.com/whitepapers/relational%20rdf%20views%20mapping.html`. Last visited July 2011.

[37] SquirrelRDF. `http://jena.sourceforge.net/SquirrelRDF/`. Last visited July 2011.