A survey of RDB to RDF translation approaches and tools

Franck Michel, Johan Montagnat, Catherine Faron Zucker

LABORATOIRE

INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR7271

A survey of RDB to RDF translation approaches and tools

Franck Michel, Johan Montagnat, Catherine Faron-Zucker

Equipes Modalis/Wimmics

Rapport de Recherche
ISRN I3S/RR 2013-04-FR

Novembre 2013 - 24 pages

Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis (I3S) - UMR7271 - UNS CNRS
2000, route des Lucioles - Les Algorithmes - bât. Euclide B 06900 Sophia Antipolis - France
http://www.i3s.unice.fr

A survey of RDB to RDF translation approaches and tools

Franck MICHEL, Johan MONTAGNAT and Catherine FARON-ZUCKER

Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France

Abstract. Relational databases scattered over the web are generally opaque to regular web crawling tools. To address this concern, many RDB-to-RDF approaches have been proposed over the last years. In this paper, we propose a detailed review of RDB-to-RDF methods and tools, considering end-to-end projects that delivered operational tools. The different approaches are classified along four major axes: motivation of the approach; mapping description language and expressiveness; mapping implementation; and data retrieval method. Then, we analyse commonalities and differences between existing approaches, and we underline common patterns in the orchestration of solutions applicable to each step of the translation process. Finally, we underline that the expressiveness of existing mapping languages is not always sufficient to produce rich semantic data and make it usable, interoperable and linkable. We therefore briefly present various strategies investigated in the literature to produce additional semantic data.

1 Introduction

Making data hosted in relational databases (RDB) accessible to the semantic web has been an active field of research during the last decade. Converting relational data into RDF or exposing relational data so that it can be queried through SPARQL[1], the query language of the semantic web, is often referred to as the "RDB-to-RDF" process. Yet, data providers willing to publish their data in a machine-readable format may be somewhat discouraged by the difficulty of choosing between the existing tools and associated mapping languages. In September 2012, the publication by the W3C of the first R2RML[2] recommendation, a standard language to describe mappings between a relational database and an equivalent RDF representation, has marked a new step towards the actualization of the web of data. R2RML encourages RDB-to-RDF tool developers to comply with a standard mapping language. On the other side, data providers should benefit of the adoption of this common language, allowing them to decouple relational data integration problems from specific tools or approaches, and ensuring sustainability. However, the choices made in the R2RML specification imply some limitations on the kinds of mappings that can be expressed. Furthermore, an implementation-independent mapping language such as R2RML does not address some of the common questions that occur when translating existing relational data into RDF, such as the choice of reusing existing vocabularies, or the way the data is accessed or queried.
Many RDB-to-RDF techniques and corresponding tools have been proposed over the last years. In spite of the observed convergence of several of them towards an R2RML implementation, the choice of an RDB-to-RDF technique and a technical implementation is not an easy task. Firstly, different techniques convey different philosophical approaches (e.g. focus on ontology learning, mapping language design, query engine design…) which have implications on the way the relational data exposed can be manipulated. Secondly, existing reviews of RDB-to-RDF approaches often provide very brief descriptions of the tools developed, making them difficult to compare. Operational questions such as the conversion of relational data into RDF repositories versus the real-time use of the native relational databases, the exposed data access and querying means, tools sustainability, etc., are hardly addressed.

1.1 Motivations for RDB-to-RDF translation

In order to grasp the diversity of RDB-to-RDF approaches, it is useful to understand the motivations of the RDB-to-RDF studies and projects, which led to the emergence of R2RML. Three common needs, that we describe below, are generally targeted: accessing data from the deep web, linking open data, and integrating multiple heterogeneous data sources.
The "deep web", as opposed to the "surface web", is a part of the web content that is hardly indexed by standard search engines. It refers to the data hidden in unstructured documents (images, scans), semi-structured documents (csv, pdf...), or structured data sources (relational databases, xml databases, NoSQL databases, LDAP directories...) only accessible through query forms but which content cannot be browsed by standard tools. As an illustration, in 2007, 70% of web sites were backed up by RDBs, which contained 500 times more data than directly available [1]. Making this huge amount of data available in a machine-readable format is expected to create opportunities for novel applications and services. In this regards, RDF is as a powerful pivot format. Yet, in order to ensure the sustainability of the applications that were developed along with the data they exploit, and to leverage the properties engineered into RDB systems over decades (scalability, ACID[3] properties, security and performance optimizations), it is preferable that the data remains hosted and delivered by the legacy RDBs,

[1] http://www.w3.org/TR/sparql11-overview/
[2] http://www.w3.org/TR/r2rml/

[3] atomicity, consistency, isolation, durability

hence the need for RDB-to-RDF techniques that can access relational data and convert it into RDF triples.

Linking open data to other related pieces of data increases its value. From this simple statement, the Linked Data principles, proposed by Tim Berners-Lee [2], recommend best practices for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF. Driven by these recommendations, the Linking Open Data[4] community project aims at extending today's web by publishing various open data sets as RDF, and setting RDF links between data items from different data sources. In other words, it intends to solve the semantic web chicken-and-egg dilemma, stating that a critical mass of machine-readable data must be available for novel mash-up applications to arise. Such applications should create added-value by repurposing data sets, using the data in some new way, possibly beyond what data providers may have initially expected. In this regard, the success of the Linking Open Data project largely depends on the accessibility of the "deep web" data, and the availability of RDB-to-RDF tools to help publish the existing relational data into RDF.

Integrating heterogeneous data has become a major challenge in several domains. In the bio-medical domain in particular, translational science refers to the continuous information exchange between research and clinics [3][4]. For instance, neurosciences have to connect, make sense of, and search across heterogeneous data and knowledge describing different organization scales (molecule, proteins, genes, cells, physiology, behaviours...) [5]. The first major step to integrating heterogeneous relational data sources is to make their semantics explicit. Relational schemas commonly provide no or poor semantics. To some limited extent, implicit semantics can be figured out from integrity constraints or usual database design patterns such as n-ary relations and inheritance. But additional semantics is frequently encoded in the application exploiting a relational database, for instance by means of domain specific rules. Moreover, relational schemas are often fine-tuned and customized for performance reasons. This results in mixing data semantics with technical concerns, making it even more difficult to figure out the original data semantics. As a result, in order to tackle the challenges of data integration in translational science, data integration techniques have to "lift" the data, that is to capture and expose its semantics in an explicit and machine-readable manner. Using RDF as a format for representing relational data appears as a powerful and promising method to achieve such data integration, in which RDB-to-RDF methods will play a key role.

[4]http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

1.2 Previous works

Facing the large variety of initiatives in the domain of relational database to RDF mapping, several studies have been conducted to compare approaches and techniques.

In 2007, the W3C decided the creation of the RDB2RDF Working Group[5] to standardize languages for mapping relational database schemas into RDF and OWL. As members of the RDB2RDF Incubator Group, Sahoo et al [6] conducted a review with a wide scope, addressing theoretical articles, proofs of concept, domain-specific projects as well as generic mapping tools. The goal of this survey was not to get into the details of each approach. Instead, it aimed at providing members of the RDB2RDF Working Group with a comprehensive overview of the different approaches that had been investigated so far, in order to serve as a basis for the definition of R2RML. Consequently, the classification they proposed is not specifically focused on mapping languages but rather explores the approaches along diverse axes: mapping creation (description language), mapping implementation (how the data is translated into RDF), query implementation (how queries are (re)written). Three additional axes are simply mentioned: mapping representation and accessibility (meta-language to express mappings), application domain, and data integration.

Hert et al [7] proposed a feature-based comparison framework that they have applied to state of the art mapping languages. The framework is derived from the use cases and requirements described by the W3C RDB2RDF working group [8]. The mapping languages are sorted into four categories: direct mapping, read-only general-purpose mapping, read-write general-purpose mapping, special-purpose mapping. This paper focuses specifically on the comparison of the mapping language features and expressiveness, but does not address the implementations proposed by their authors or the way queries are rewritten.

Also to be mentioned, Sequeda [9] proposed a short high-level review of major RDB-to-RDF mapping products, along with a brief description of each of them. Unlike the two previous reviews that almost exclusively focus on academic works, this paper reviews products either from the academic world or from the industry. As a result, although very succinct, this work is an interesting entry point to broaden the scope towards the industrial approaches.

Sequeda et al [10] also surveyed methods that apply Direct Mapping principles to automatically translate a relational database into RDF. They studied methods proposed in the literature to extract ontological knowledge as RDFS or OWL from SQL DDL (Data Description Language) representations. This ranges from simple approaches (table to class, column to property) to more advanced ones that try to discover relations such as many-to-many, subsumption, has-part/is-a-part-of, symmetric and transitive relations, and

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

SQL features such as integrity constraints, checks and triggers. Finally, authors propose a consolidated approach, based on the work of Tirmizi et al [11], that exploits all the possible combinations of primary and foreign keys in relational tables, in which they try to gather the best of all studied approaches. As a conclusion, authors acknowledge that the quality of an ontology resulting of a direct mapping highly depends on the richness of the SQL schema with respect to its encoding of domain semantics.

1.3 Goal of this work

In this paper, we propose a detailed review of state of the art RDB-to-RDF tools, either academic or industrial. We make a specific focus on the specificities of each technique, and we describe, as much as possible, the capabilities of the different tools studied. Taking into account practical concerns, we only consider end-to-end projects that delivered operational tools to implement the RDB-to-RDF process, excluding proofs of concepts and early prototypes. A classification is proposed to identify the different approaches along four major axes that span from the motivations to the mapping description and service deployment concerns: (i) motivation of the approach; (ii) type of mapping, mapping description and expressiveness; (iii) mapping implementation (how and when the data is converted into RDF), and (iv) data retrieval method (query-based, linked-data, graph dump). Also, when available, we provide information regarding the project maturity and sustainability.

2 Classification of RDB-TO-RDF Approaches

2.1 Motivation of the approach

In the literature, the many projects, prototypes or articles dealing with the RDF access to relational databases consider the subject from somewhat varying perspectives, depending on their initial motivations. We can roughly sort them out into three categories that we describe below. Any approach may fall in one or several, or even all of the three categories.

Ontology learning
Ontology learning approaches are focused on the engineering of ontologies, their goal is generally to extract ontological concepts and relations from relational schemas and data. The scope may vary from simple approaches reflecting the relational schema (using a simple direct mapping method to build an ad-hoc ontology), to more complex methods looking for specific database design patterns such as many-to-many relations and concept subsumption relations. Some methods propose to analyse both the relational schema and data using data mining techniques in order to figure out new ontological concepts and properties. For instance, data redundancy may evidence categorization patterns indicating subsumption relations. Some semi-automatic approaches iteratively create the ontology, suggesting probable matches between the database and an existing ontology, or extending an existing ontology with additional concepts and properties learnt from the database schema.
Ontology learning approaches are often accompanied by the definition of a mapping language, and optionally the implementation of a proof-of-concept or prototype.

Definition of a generic-purpose mapping language
The goal of such approaches is basically to define a mapping language able to describe different situations of mapping between a relational database and an RDF representation. Unlike the ontology learning domain, the ability to describe a mapping to an existing ontology will be a key concern here. More generally, the primary goal is to ensure that the expressiveness of the mapping language spans a large scope of situations, from a simple direct mapping to complex cases where the similarity between the database and the target ontology is low. Some approaches even address the mapping of databases lightly structured or not in first normal form. Some mapping languages provide the ability to define complex transformations making use of different methods such as regular expression matching, keyword search or natural language processing.

Implementation of an RDB-to-RDF transformation engine
Most approaches, either focused on ontology learning or on the definition of a mapping language, propose the implementation of a transformation engine to validate and showcase the results of the approach. This may be implemented as either a query processing engine typically able to process SPARQL queries, or the transformation of the whole relational data into an RDF representation at once. Anyway, some RDB-to-RDF approaches more specifically target the development of a production-class query engine, that is, a query engine able to handle a large number of concurrent requests while ensuring acceptable performances, and handle complex queries with acceptable response time. Those approaches are concerned with the way a (SPARQL) query to the target data model will be translated into the source database query language (SQL query rewriting), and by the efficient execution of the translated query (query optimization and query planning).

2.2 Mapping description

The mapping description refers to the way the mapping between the relational database and an RDF representation is described. It is generally driven by the fact that the resulting mapping should either (i) come up with an ad-hoc ontology that reflects the relational schema, or (ii) comply with existing well-defined domain-specific semantics, by reusing domain ontologies and possibly entailing more complex mappings. The first is referred to as the *Direct Mapping*, while the latter is called *Domain Semantics-Driven Mapping*.

In the literature, *Direct Mapping* is frequently used as a synonym of *Automatic Mapping*, and *Manual Mapping* as a synonym of *Domain Semantics-Driven Mapping,* although this happens to be misleading: a direct mapping is generally created automatically and later customized manually. Yet, despite the manual edition, it can remain nothing more than a direct mapping. Conversely, the domain semantics-driven mapping is often called manual mapping, although an automatically generated direct mapping is frequently used as a starting point to more complex domain-specific mappings. In addition, some approaches attempt to automatically discover mappings far beyond the definition of direct mappings. The latter could be called "automatic discovery of domain-semantics mapping". Consequently, hereafter we shall use the terms *Direct Mapping* or *Domain Semantics-Driven Mapping*. The terms *manual* and *automatic* will be used for exactly what they mean: the manual edition of a mapping by a human agent, vs. the automatic generation of a mapping by a program.

Direct mapping (also known as automatic mapping, local ontology mapping, table-to-class)
The direct mapping approach intends to convert relational data into RDF in a straightforward manner, making the process simple. It consists in the automatic creation of URIs following simple rules defined by Tim Berners-Lee [12]:

- table-to-class: a table is an ontological concept with URI "namespace/database/table";
- column-to-property: each column of a table is an ontological property with URI "namespace /database/table/column";
- row-to-resource: each row of a table is a resource, i.e. an individual which class is represented by the table. The resource URI is formed using the primary key: "namespace/database/table/primaryKey" or "namespace/database/table#primaryKey";
- cell-to-literal-value: each cell with a literal value is the object of a data property;
- cell-to-resource-URI: each cell with a foreign key constraint is turned into the object of an object property.

Applying this set of rules automatically creates an ad-hoc RDF vocabulary *reflecting exactly the structure of the relational schema*. In ontology learning approaches, the term *local ontology mapping* is sometimes used to refer to ad-hoc ontologies created by the direct mapping process. To avoid exposing unnecessary or sensitive data such as passwords, most approaches automatically generate a first mapping that can be manually customized to some extent. Some specific cases such as multi-column primary keys and tables with no primary key are also generally addressed. Optionally, the direct mapping can be improved by the automatic detection of common database design patterns: many-to-many relations suggested by join tables (tables which all columns are foreign keys to other tables), and implicit subclass relationships suggested by a primary key used as a foreign key [13]. In the latter case, the literature argues that, in the context of databases not in the third normal form, this pattern may reveal a vertical partitioning (that consists in splitting a table into several smaller tables for performance concerns) rather than a subsumption relationship [11][10]. Yet, open research questions remain with regards to the possibility of translating relational database triggers into additional knowledge in the form of semantic rules.
The Direct Mapping method typically applies when no domain ontology exists to which the relational schema could be mapped, or when the goal is to rapidly make data sources available in a machine-readable format, with few concerns for semantic interoperability. Direct Mapping can also address versatile environments in which databases may appear and disappear frequently with no time for manual alignment [14]. When semantic interoperability is required, ontology alignment methods are used later on to align the local ontology with existing domain ontologies.
The W3C has proposed a specification "A Direct Mapping of Relational Data to RDF" that specifies good practices of the direct mapping approach [15]. It is a very simple version of the direct mapping, essentially the formalization of the rules enounced by Tim Berners-Lee.

Domain Semantics-Driven mapping (also known as manual mapping, transformative mapping)
Although the direct mapping approach transforms the content of a relational database into RDF, it is unable to capture the true semantics of the data. At best, this semantics is implicitly suggested by the relational schema metadata. At worse it consists of domain-specific rules encoded in the application that exploits the database. This is outlined in a short but enlighten feedback from the Ordnance Survey of Great Britain: "*existing tools designed to generate ontologies based on database schemas are missing the point: databases are rarely good descriptions of a domain, being the result of both performance optimisation processes and contingent maintenance history. And, in any case, the schema itself will not support a full description of the domain, other relevant relationships often being buried in code or in the encoding of various attributes*." [16].
To overcome these limitations, the *domain semantics-driven mapping* approach applies when the relational database must be translated using concepts and properties of existing ontologies. The database and the ontology may have been designed separately, hence the similarity level between them may be low. A typical use case is the alignment of a legacy database with an existing ontology that refers to or describes the same domain of interest.
The functionalities of domain semantics-driven systems allow to describe highly expressive mappings, able to bridge the conceptual gap between RDB and RDF. Table 1 lists the features that may be supported.

Feature name	Feature description
generation of user defined unique Ids	Ability to generate URIs of resources beyond the simple use of primary key values: reusing and combining column values, allowing for conversion tables, etc.
logical table	Ability to read tuples not only from tables but also from SQL views or from the result of an SQL query.
column selection (also called projection)	Ability to select only a subset of the columns of a table to translate. This is a very basic feature, almost a minimum pre-requisite of any RDB-to-RDF tool.
column renaming	Ability to map a column to an RDF property with a different name. This is not always possible in a direct mapping but quite obvious in a domain semantics-driven mapping.
select conditions	Ability to translate only a subset of the tuples of a table using a select-where condition
vocabulary reuse	Ability to map relational entities to instances of existing vocabularies and ontologies. This is the main difference between domain semantics-driven mapping and direct mapping approaches.
1 table to n classes	Ability to use the values of a column as a categorization pattern: tuples of the table will be translated into instances of different ontological classes based on the value of this attribute. This feature can be seen as an extension of the "select conditions" feature as it results in not only filtering out rows, but the filter helps selecting rows to be converted into instance of one class or another.
many-to-many relation to simple triples	Many-to-many relations are usually implemented in relational databases as a join table which columns are all foreign keys to other tables (n-ary relations). This feature consists in the ability to translate many-to-many join tables into simple triples. This opposes to a basic direct mapping in which the join table will be translated into a distinct class.
blank nodes	Ability to generate blank nodes and refer to them within the graph produced during the translation process. Blank nodes can be used for instance to translate a table without a primary key.
data types	Ability to handle relational data types consistently with RDF data types per SQL-XSD mapping.
data transformation	Ability to apply transformation functions to the values before generating the RDF triples. This can be used to perform complex type conversion, compute a value using several columns, and applying methods such as string manipulation functions, decimals type conversions, etc.
named graphs	Ability to create not only a default RDF graph but also multiple named graphs within a single mapping definition.
user-defined namespaces	Ability to declare and use namespace prefixes.
static metadata	Ability to attach static metadata (such as licensing or provenance information) to the produced graphs, and possibly to all RDF entities or instances of a certain class.

Table 1: Possible features of mapping languages

To support domain semantics-driven mappings, mapping description languages generally implement two different strategies:

(i) The mapping description essentially relies on SQL queries to present the data as expected. The expressiveness is therefore constrained by that of SQL and complex cases that outreach the expressiveness of SQL cannot be addressed (unless supported by extensions of a specific RDBMS). On the other hand, the rewriting of a query on the target RDF data (generally a SPARQL query [17]) into SQL is almost straightforward. This allows developers to tune queries and rely on the database system performance. Besides, the fact that SQL is widely known facilitates the adoption of the mapping language by data providers who hardly need to learn a new mapping language.

(ii) The mapping description uses a specific dedicated language. A query on the target RDF data will be rewritten into SQL queries. This approach is not constrained by the expressiveness of SQL. As a result it can be extended in order to meet specific complex needs such as keyword search, regular expression matching, natural language processing, data mining, etc. Nevertheless, it must be noticed that most existing projects hardly reach the expressiveness of SQL (for instance, aggregation or groupings are not always possible although they are natively supported by SQL).

Some mapping languages such as R2RML and D2RQ use both strategies simultaneously: they are able to complement SQL snippets with specific mapping descriptors.

2.3 Mapping implementation

Given a mapping description, that is the set of rules that map a relational model to a target ontology, the mapping implementation refers to the way database tuples are translated into ontological instances (individuals). Two methods can be applied: the *data materialisation*, or the *on-demand mapping*.

Data materialisation

The *data materialisation* approach is the static transformation of the source database into an RDF representation, in the manner of the data warehouse approaches. It consists in the application of the mapping rules to the whole content of the database to create an equivalent RDF graph. For this reason, the literature also refers to it as "graph dump", "graph extraction" or "RDF dump". When the materialisation process completes, the resulting RDF graph can be loaded into a triple store and accessed for instance through a SPARQL query engine. This whole process is often referred to as the *Extract-Transform-Load* (ETL) approach, that conveys the idea of data materialisation and loading into a triple store.

With this approach, a large number of data sources can be integrated, in the limit of the triple store and query engine capacity. Besides, a major advantage of the materialisation is to facilitate further processing, analysis or reasoning over the RDF data, including the execution of heavy inference rules. Indeed, as the RDF data is made available at once by the materialisation process, third party reasoning tools can be used to apply complex entailments. Later on, complex queries can be answered without compromising run-time performances since the complex reasoning tasks have been performed at an earlier stage.

Several limitations are to be noticed though. This solution hardly supports very large data sets, as the size of the graph produced may exceed memory capacity. Another limitation concerns the way to deal with outdated data: in the context of an application that updates the relational data frequently, the materialized RDF graph may be rapidly outdated. To address this issue, the extraction process may be run periodically. In this case however, a compromise should be found between the cost of materializing and reloading the graph, and the degree of tolerance of the application to outdated data.

On-demand mapping

Conversely to the data materialisation method, the *on-demand mapping* approach consists in the run time evaluation of queries against the relational data. In this model, the data remains located in the legacy database. Whatever the way the converted data is accessed, queries to the target RDF data must be rewritten into SQL at query evaluation time.

The advantages and drawbacks of this approach are the opposite of the materialisation approach. It is well suited in the context of very large data sets that would hardly support centralization due to resource limitations. It guarantees that the data returned is always up to date since no copy of the data is done. Besides, it easily allows for the enforcement of access control policies implied by privacy or confidentiality constraints such as in the case of medical data.

On the other hand, query performances can be severely penalised if entailment regimes must be implemented [6], or if many data sources are to be integrated together. It is suggested that in some cases, the expressiveness of SPARQL queries should be limited to be processed by on-demand mapping systems: in particular, a variable in predicate position (resourceA ?r resourceB) or a variable in object position representing a class (resourceA rdf:type ?c) lead to "union bomb" issues [18][19].

2.4 Data retrieval (query implementation)

Independently of the way a mapping is implemented, the RDF data can be retrieved using several methods: by sending a query to a query processing engine, using the linked data paradigm, or in a bulk manner by a graph dump operation. The choice of the method depends largely on how the data should be exploited. Below we briefly describe those methods.

Query-based access

The RDF data is retrieved by means of a query, generally expressed in SPARQL, the standard language to express queries over RDF data. It must be mentioned that, although marginal, a few tools implement their own query language. A SPARQL query processor can be used to run a SPARQL query against an RDF repository (in the data materialisation approach), or against a relational database (in the on-demand mapping approach). In the former case, a SPARQL query processor can be accessed through an HTTP server using the SPARQL protocol [20]: this results in a SPARQL endpoint. In the latter case, the SPARQL query must be rewritten into SQL and, conversely, SQL results must be translated into equivalent SPARQL results according to the mapping.

Besides SPARQL, early approaches proposed other query languages, nevertheless the standardization of SPARQL has deprecated those initiatives. Additionally, some RDBS providers have proposed alternative solutions by integrating a query evaluation engine within the native RDBS evaluation engine. This is the case of SPASQL that allows for the execution of SPARQL queries within SQL statements.

Entity-level access (Linked data)

Each logical relational entity translated into RDF is assigned a unique URI that identifies it in the data graph. According to the principles of the Linked Data [2], it should be possible to *dereference* any such URI by performing an HTTP GET method on this URI, as if it was a URL. The result of this operation will be a representation of the entity identified by the URI. The output format of the data description is generally agreed

during a usual HTTP content type negotiation procedure between the client and the web server.
Two access methods are advised in [21], in relation with good practices to define URIs:

- For an informational resource, a simple web lookup with HTTP content negotiation will return a representation of the entity (in XHTML, RDF/XML, N3, JSON, etc.).
- If the URI refers to a non-informational resource, i.e. an abstract concept or a physical object, it should not be dereferenced directly (if one dereferences a URI to a person, the physical person itself is not retrieved, instead a document that relates to the person is expected). An HTTP GET to such a URI should return HTTP status *303 See other*, providing the URI of an informational resource that relates to or is a representation of the non-informational resource: e.g. an HTML document describing a person.

This recommendation can somehow be relaxed due to technical concerns like the network traffic, time consuming definition of multiple URIs, etc.
The term *Linked Data* also sometimes refers to the ability to dereference not only a resource URI, but also a URI to a logical entity of the source database. For instance, dereferencing the URI of a database will return a human-readable list of classes corresponding to the tables, dereferencing the URI of a class will return URIs of instances of the class. Closer to the spirit of the Linked Data, D2RQ will dereference a class URI by providing a few triples stating that it is a database or a class, including *rdfs:seeAlso* statements when relevant. Such methods can be exploited by crawlers of external search engines to index the content of the database.

Graph dump access
In the graph dump access method, the client performs an HTTP GET request to the database URI and retrieves the entire RDF graph at once. This method is most likely available with data materialisation approaches, in which the graph has been materialized beforehand. It is generally not available in on-demand mapping approaches as building large graphs on demand (at run time) may induce significant performance issues.
The "SPARQL 1.1 Graph Store HTTP Protocol" [22] can be seen as an attempt to standardize the graph dump access mode, while extending it to a larger scope of graph management methods.

3 R2RML

R2RML [23] is a generic-purpose language targeted to describe a set of mappings that translate data from a relational database into an RDF representation. It is the result of a long process and preliminary works held by the W3C. During the W3C Workshop on RDF Access to Relational Databases[6] in 2007, Dean [24] suggested a set of features that a standard mapping language should support. The outcome of this workshop was the formation of the W3C RDB2RDF Incubator Group[7] (abbreviated XG). The Incubator Group ran a comprehensive survey of existing projects and approaches [6], and described high level recommendations of what aspects a recommendation to map relational databases to RDF should cover [25]. It concluded with the recommendation to create the RDB2RDF Work Group to standardize a mapping language. Here we present the results of the group.

3.1 Features

The W3C RDB2RDF Work Group proposed a list of 11 requirements, expressed as features that MUST (mandatory) or SHOULD (optional) be provided by R2RML [8]. Refer to Table 1 for a detailed description of each feature below.
Mandatory features state that R2RML must (i) support both the direct mapping and domain semantics-driven mapping (called transformative mapping); (ii) provide sufficient information for a processor to support both the on-demand mapping (rewrite SPARQL queries into SQL), and the data materialisation. Other mandatory features are: generation of globally unique identifiers, support data types, SQL column renaming, many-to-many relation to simple triples, 1 table to n classes.
Optional features are: data transformation, named graphs, namespace declaration, static metadata. Auer et al [8] also describe an "update logs" "nice-to-have feature": the mapping should provide extension points to support the creation of update logs of relational data. It is nevertheless mentioned that this is out of the scope of R2RML.
The final R2RML recommendation includes all mandatory features listed above, and all optional features with the exception of the data transformation: data transformation is left to the (limited) capabilities of SQL in terms of string or number manipulations, but no complex data transformation method is specified.
Furthermore, Table 1 describes three features that were not explicitly addressed by Auer et al, but that are part of R2RML: select conditions, column selection and blank nodes. The "select conditions" feature is implicit in R2RML as it is a pre-requisite of the "1 table to n classes" feature. Similarly, the "column selection" feature is implied by the more global support of the transformative mapping. Lastly, blank nodes are implemented in the definition of R2RML RDF terms, making them natively supported in R2RML.
Finally, here is a summary of the features supported by R2RML: user-defined URIs, logical table, column selection, column renaming, select conditions, vocabulary reuse, 1 table to n classes, many-to-many relation to simple triples, blank nodes, data types, vendor specific data types, named graphs, user-defined namespaces, static metadata. No support of data transformation is specified in R2RML, instead it relies on SQL capabilities to perform transformations.

[6] http://www.w3.org/2007/03/RdfRDB/

[7] http://www.w3.org/2005/Incubator/rdb2rdf/

3.2 Mapping description

The recommendation specifies that an R2RML processor MAY include an R2RML default mapping generator that generates an R2RML mapping, possibly in the form of an R2RML mapping document, intended for further customization by a mapping author. The default mapping must comply with direct mapping rules specified in the R2RML companion document: "A Direct Mapping of Relational Data to RDF" [15].
The mapping definition language is RDF with Turtle syntax. An R2RML mapping document is called an *R2RML mapping graph*. The language also embeds SQL requests making use of any SQL construct (select tables or views, aggregation). A mapping consists of several *triples map*, each triple map specifying how to map each row in a logical table of the input relational database to a number of RDF triples. The logical table may be a table, an existing SQL view, or the result of a valid SQL query to be executed over the input relational database.
A triple map has two main parts: (i) a *subject map* that generates the subject of all RDF triples that will be generated from a logical table row. The subjects often are IRIs generated from the primary key column(s) of the table; (ii) multiple *predicate-object maps* that in turn consist of *predicate maps* and *object maps*. Triples are produced by combining the subject map with a predicate map and object map, and applying these three to each logical table row. By default, all RDF triples are in the default graph of the output dataset. A triple map can contain *graph maps* that place some or all of the triples into named graphs instead.

3.3 Implementations

The first R2RML recommendation was issued Sept. 27th 2012. Several candidate implementations have been evaluated against compliance tests described in the *R2RML and Direct Mapping Test Cases*[8], which results are reported in the *RDB2RDF Implementation Report*[9]. The test cases address both the W3C Direct Mapping recommendation and R2RML. The W3C Direct Mapping recommendation describes very basic direct mapping rules, with somehow limited interest. As a result, in the following, we have not considered implementations which support is limited to the W3C Direct Mapping.
The R2RML implementations considered are: **DB2Triples** (passed all R2RML tests but one); **OpenLink Virtuoso** (29 R2RML tests out of 62 in status "cannotTell", i.e. test cannot be run. no reason is given); **Morph** (failed on a few R2RML tests but fair support anyway); **RDF-RDB2RDF (**11 failed tests out of 62); **Ultrawrap** (passed all R2RML tests); **XSPARQL** (passed all R2RML tests).
In addition, **Oracle Spatial and Graph 12c**, released in July 2013, supports R2RML. At the time of writing, it has not been officially tested against the R2RML test cases and thus is not yet mentioned in the RDB2RDF Implementation Report.

[8] http://www.w3.org/TR/2012/NOTE-rdb2rdf-test-cases-20120814/
[9] http://www.w3.org/TR/2012/NOTE-rdb2rdf-implementations-20120814/

4 RDB-to-RDF tools

This section describes the tools studied in the context of this paper. An explanation of how the classification introduced in section 2 applies to each tool is given. Section 4.1 gathers the tools that comply with R2RML section 4.2 the tools that propose their own mapping language. In each section, the tools are listed by alphabetical order. Table 2 (section 4.3) provides a synthetic summary of the information provided here after.

4.1 R2RML-compliant tools

4.1.1 *DB2Triples*

DB2Triples[10,11] is an implementation of the W3C R2RML and Direct Mapping recommendations. It is developed by the company Antidot[12] as part of a larger software suite. DB2Triples is delivered as a java library, available under the terms of the LGPL open source licence, and validated with MySQL and PostgreSQL back-ends. It takes as input an R2RML document, a database connection and a SPARQL query, and returns the results in RDF/XML, N3, N-Triples or Turtle. Therefore, it is able to process SPARQL queries, but it is not a SPARQL endpoint able to receive requests over HTTP.
Main scope: RDB-to-RDF query-based transformation engine using the R2RML mapping description.
Mapping description: see R2RML.
Mapping implementation: on-demand.
Mapping language features: see R2RML. Passed all R2RML test cases but one.
Data retrieval: SPARQL query-based.
Sustainability: DB2triples version 0.9.9 is compatible with the R2RML[13], and the Direct Mapping[14] Working Drafts, and successfully passed the *R2RML and Direct Mapping Test Cases*.

4.1.2 *Morph*

Morph[15] is an R2RML mapping processor, developed in Scala, by the developers of R_2O and ODEMapster. The development is ongoing, and only few documentation is available so far.
Main scope: RDB-to-RDF transformation engine using the R2RML mapping description.
Mapping description: domain semantics-driven mapping (R2RML).

[10] http://www.antidot.net/fr/Actualites/Produit/Antidot-fournit-db2triples-en-Open-Source
[11] https://github.com/antidot/db2triples
[12] http://www.antidot.net/
[13] http://www.w3.org/TR/2012/WD-r2rml-20120529/
[14] http://www.w3.org/TR/2012/PR-rdb-direct-mapping-20120814/
[15] https://github.com/jpcik/morph

Mapping implementation: on-demand, data materialisation (called batch-upgrade mode).
Mapping language features: see R2RML. Morph does not pass 8 of the 62 R2RML test cases. *Not supported features*: blank nodes, named graphs in on-demand mapping (but supported in data materialisation), static metadata.
Data retrieval: SPARQL query-based. The data materialisation also allows for the ETL approach.
Sustainability: developers argue that they intend to continue the support and evolution of Morph in the middle-term, possibly extending it further than the official R2RML specification like supporting Google Fusion Tables.

4.1.3 *Oracle Database 12c*

Oracle Spatial and Graph[16] (formerly Oracle Semantic Technologies) [26][27] is an option of *Oracle Database Enterprise Edition* (EE). Version 12c, released in July 2013, comes with the *RDF Semantic Graph* data management and analysis features, that support RDB-to-RDF conversion.
RDF Semantic Graph mainly focuses on the storage, simultaneous querying and reasoning on relational and RDF data. The RDF graph store can scale up to billions of triples, supports bulk and incremental load, graph versioning, and the semantic indexing of documents. RDF querying is possible using SPARQL 1.1. SPARQL graph patterns can be included within an SQL query in order to join RDF and relational data. RDFS/OWL2 (RL/EL profiles) reasoning is supported (OWLSIF, OWLPRIME, RDFS++) simultaneously on RDF and relational data, along with inference proofs and explanations, parallel inferencing, user-defined rules. The architecture allows for using the native reasoner or a 3rd party one like PelletDB. The security model allows to enforce restrictions at different levels, from the graph to the triple.
RDB-to-RDF conversion is supported since version 12c by providing RDF views on relational tables, SQL views, and SQL query results. W3C Direct Mapping and R2RML are supported. The RDF view can be queried through SPARQL 1.1.
Oracle Spatial and Graph exploits several enterprise features such as the table compression (optimize disk space and memory usage), the partitioning option (performance, scalability), and the *Real Applications Clusters* (availability, scalability). As a result, using Oracle Spatial and Graph requires to acquire licenses for Oracle Database EE and Partitioning option.
Main scope: RDB-to-RDF query-based transformation engine, using the R2RML mapping description.
Mapping description: domain semantics-driven mapping, support of R2RML, automatic generation of a direct mapping compliant with the W3C recommendation.
Mapping implementation: on-demand mapping.
Mapping language features: support of W3C's Direct Mapping and R2RML. However, no result is available so far as to the conformance tests. Consequently, Table 2 shows interrogation marks for all the features, although we can assume that at least most common features are supported.
Data retrieval: SPARQL query-based.
Sustainability: RDB-to-RDF is newly supported in release 12c. No information is available on its support by Oracle in the future.

4.1.4 *RDF-RDB2RDF*

RDF-RDB2RDF is a Perl open source software based on the *Perl RDF* library, delivered under the GPL license. Perl RDF is a very complete library including the following features: RDF store in memory, support for relational databases (MySQL, PostgreSQL) and SQLite, SPARQL 1.1 query processor and endpoint, Linked Data server, RDFa parser, WebID (FOAF+SSL) + ACLs, GRDDL, Microformats, HTML5.
Main scope: RDB-to-RDF materialisation-based transformation engine, using the R2RML mapping description.
Mapping description: domain semantics-driven mapping (R2RML).
Mapping implementation: data materialisation.
Mapping language features: R2RML. RDF-RDB2RDF does not pass 7 of the 62 R2RML test cases with PostgreSQL.
Data retrieval: the materialized data cannot be retrieved immediately. However it must be noted that Perl RDF provides a SPARQL endpoint in which the RDF data can be loaded, thus falling in the ETL approach.
Sustainability: last update Sep. 2013.

4.1.5 *Ultrawrap*

Initially developed by the University of Texas in Austin, Ultrawrap[17] is now a commercial product produced by company Capsenta, founded in 2011 as a spin out of the University of Texas. It is based on the use of SQL views to present the relational data as triples, thus making the query rewriting process almost straightforward, while relying on the native SQL optimization engine.
Ultrawrap automatically generates a local ontology following the direct mapping W3C's recommendation. The triple representation of the relational data is implemented as a three-column SQL view (subject, predicate, object), that consists in the union of all the queries that define all the triples as defined by the local ontology [28]. Consequently, a SPARQL query can be naively rewritten into an SQL query on the SQL view, benefitting from the native query optimizer of the relational database. Advantage: SPARQL execution as fast as SQL.
Recently, support for R2RML and D2RQ mapping languages has been added. There is no description,

[16] http://www.oracle.com/technetwork/database-options/spatialandgraph/overview/rdfsemantic-graph-1902016.html

[17] http://capsenta.com/ultrawrap

however, of the way the mapping description is derived into the SQL view. We can assume that an R2RML/D2RQ document is "compiled" into an SQL view that will reflect each triple map. To be mentioned that the support of R2RML named graphs does not seem easy using only the SQL triple view. A GUI that is part of the tool suite helps align the local ontology onto a domain ontology.
Ultrawrap was released to first beta customers in May 2012. Prices are available on demand only.
Main scope: RDB-to-RDF query-based transformation engine using the R2RML mapping description.
Mapping description: domain semantics-driven mapping, support of R2RML and D2RQ mapping documents. Automatic generation of a direct mapping.
Mapping implementation: on-demand, execution of SQL views.
Mapping language features: see R2RML. Passed all R2RML test cases.
Data retrieval: SPARQL query-based, Linked Data.
Sustainability: first beta release delivered in May 2012, no new public release since then according the company's web site.

4.1.6 *Virtuoso Universal Server & Virtuoso's RDF Views*

Virtuoso Universal Server[18] is a commercial and open-source object-relational database system developed by OpenLink. It is targeted to meet enterprise needs regarding data management, access and integration. This is a very comprehensive tool suite, that comes with production-class features like a relational database, clustering, data replication, miscellaneous security options, RDF triple store, reasoning capabilities (OWL subset), multiple data sources integration (SQL, RDF, XML, free text, CMS, aggregation feeds...), Web Application Server. The *RDF Views of SQL data*[19] functionality is part of the RDF and SPARQL tool suite, and makes it possible to expose relational data as RDF.
The Virtuoso Open-Source Edition[20] is a sub-set of the Universal Server. Limitations of the open source edition concern production-class features such as the clustering and data replication functions. The RDF Views open source edition only supports the Virtuoso-based relational database whereas the commercial edition supports most well known relational database systems. The open source edition provides the following features: Object-Relational Database for SQL, XML, RDF, and Free Text; RDF store and SPARQL end-point (including RDF Views); WebDAV Server; Web Services Platform for SOA; Web Application Server (see the full feature list[21]).

Main scope: enterprise data integration, RDB-to-RDF query-based transformation engine using the R2RML mapping description.
Mapping description:
- Manual domain semantics-driven mapping using the declarative *Meta Schema Language*[22] (MSL). The language is an extension of the SPARQL query language, meshed with Virtuoso's SPASQL (SPARQL-inside-SQL) functionality. The mapping does not only consist in an MSL document, but it is itself a repository in which mapping patterns can be grouped in named sets, named sets can then be managed using operations such as create, drop, alter, etc. The rich web interface provides a wizard to automatically generate a direct mapping (limited to locally-stored Virtuoso-based relational database in the open source version).
- R2RML support is achieved by the inclusion of a simple adaptor which basically translates R2RML syntax to Virtuoso's own Linked Data Views syntax, which can then be executed to create the Linked Data Views themselves.
Mapping implementation: on-demand. The data materialisation is possible but not that easy, and this is definitely not the intended goal as Virtuoso comes with its own SPARQL endpoint.
Mapping language features: see R2RML. 29 R2RML test cases (out of 62) are in status "cannotTell", i.e. test cannot be run. No explanation why.
Data retrieval:
- Query-based through a SPARQL 1.1 endpoint, support for OpenID, OAuth and WebID authentication options. SPASQL is also supported, that is, the ability to include SPARQL statements into SQL requests.
- Linked Data: expose any Virtuoso-housed data (only solution provided in the open source version), HTTP-, ODBC-, JDBC-, and otherwise-accessible SQL and XML data sources as URI dereferenceable linked RDF data (Linked data browser). Data retrieved as RDF/XML, JSON, N3.
Sustainability: actively maintained by the company OpenLink. Note that the DBpedia project is operated using a Virtuoso triple store.

4.1.7 *XSPARQL*

XSPARQL[23] is a query language combining XQuery and SPARQL for transformations between RDF and XML in either direction. RDF/XML provides a lot of flexibility in how one and the same RDF graph can be serialized, thus abstracting away from a fixed tree-like structure. This results in potentially lots of different XML serializations of the same graph, bearing the same semantics. This makes complicated and tedious the use of XSLT+XPath processors that treat RDF/XML as XML data and not as a set of triples. To address this difficulty, XSPARQL merges SPARQL components into XQuery FLWOR expressions. It is typically

[18] http://www.w3.org/wiki/VirtuosoUniversalServer
[19] http://virtuoso.openlinksw.com/whitepapers/relational%20rdf%20views%20mapping.html
[20] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSIndex
[21] http://virtuoso.openlinksw.com/features-comparison-matrix/
[22] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSSQL2RDF
[23] http://xsparql.deri.org/

designed to address issues such as extracting RDF data out of existing (X)HTML Web pages, allowing an RDF-based client software to communicate with XML-based Web services, or even enriching an RDF graph with deduction rules described as RDF-to-RDF mapping.
More recently, Lopes et al. [29] have defined an RDB-to-RDF extension to XSPARQL that integrates a subset of SQL (select-from-where) within XSPARQL. This language extension provides the ability to add SQL snippets embedded in XQuery FLWOR expressions to query the relational data, and transform it into XML or RDF. The RDF data is generated by means of a specific "construct" expression with nested XQuery FLWOR expressions. On top of this, an R2RML mapping document can be interpreted to produce appropriate "construct" expressions that in turn materialize the RDF data. The originality of this approach is that it did not require the development of a specific module in the XSPARQL engine. Instead, the whole process is totally described using only the XSPARQL language: the R2RML mapping document is read as input RDF data, XQuery FLWOR expressions are used to parse it and query the database accordingly, and ultimately the RDF data is generated using "construct" expressions.
R2RML is supported with the exception of named graphs, and the fact that a TriplesMap cannot be expressed as the result of an SQL query (only relational tables and views are supported).The RDF representation of the database is produced in the manner of a data materialisation. Supporting R2RML using the on-demand mapping implementation should be possible but not straightforward: it would first require to design a query rewriter to convert a SPARQL query into an XSPARQL query that would in turn make the appropriate SQL queries.
XSPARQL comes with a prototype implementation of the XSPARQL language developed in Java. It is distributed under the Apache 2 open source license.
Main scope: generic-purpose mapping language, RDB-to-RDF materialisation-based transformation engine.
Mapping description: domain semantics-driven mapping (R2RML).
Mapping implementation: data materialisation.
Mapping language features: see R2RML. Passed all R2RML test cases, however logical tables and named graphs are not supported.
Data retrieval: SPARQL-query based.
Sustainability: version 0.5 released in November 2012, supporting both Direct Mapping and R2RML. XSPARQL passed all R2RML and DM test cases.

4.2 Non-R2RML tools

4.2.1 *Asio Semantic Bridge for Relational Databases and Automapper:*

Asio Tool Suite[24] is a commercial product developed by BBN Technologies. It provides the following tools: *Asio Semantic Query Decomposition* (SQD), *Asio Semantic Bridge for Relational Databases*[25] (SBRD), *Asio Semantic Bridge for Web Services*, and two open source tools: *Parliament* and *Snoggle*.
Asio SBRD is the component that performs RDB-to-RDF conversion: it translates SPARQL queries into SQL and returns the results in RDF, by applying query planning and optimization techniques. The BBN's Automapper tool applies D2RQ-based direct mapping rules to create the local ontology that describes the relational schema.
The Asio SQD module addresses the federation of multiple data sources: it breaks down SPARQL queries expressed using a domain ontology into sub-queries using the local ontology of each data source, and distributes these optimized sub-queries to the applicable data source: relational database, web service, triple store. Mapping rules from the local ontology to the defined domain ontology are described as a set of SWRL[26] rules.
Main scope: enterprise data integration, RDB-to-RDF query-based transformation engine.
Mapping description: domain semantics-driven. A direct mapping is initially created based on D2RQ tools, and augmented with OWL property restrictions to model data types and nullable/not nullable properties. Additional mapping rules between the domain ontology and the local ontology are created in SWRL.
Mapping implementation: on-demand mapping.
Mapping language features: n.a.
Data retrieval: SPARQL query-based.

4.2.2 *D2R Server and the D2RQ language*

The D2R Server[27] [30] [31] is an open source academic project. It provides an integrated environment with multiple options to access relational data using different methods such as the SPARQL endpoint, Linked Data (content negotiation, HTTP 303 dereferencing), RDF dump, and Jena API based access (API calls are rewritten to SQL).
Main scope: generic-purpose mapping language, RDB-to-RDF query-based transformation engine.
Mapping description: D2RQ supports both the direct and domain semantics-driven mappings. The DR2Q declarative mapping language [32] is formally defined by an RDFS schema. It is the successor to the XML-based D2R MAP language [33]. The mappings are expressed in RDF, but also largely rely on SQL fragments to express select conditions or to use aggregate functions. Existing ontologies (RDFS/OWL) can be reused in order to incorporate domain semantics in the mapping process. The automatic mapping generates a D2RQ direct mapping that reflects the database schema, thus creating a local ontology. This direct mapping can be customized manually. Optionally, the direct mapping generated can follow the

[24] http://bbn.com/technology/knowledge/asio_tool_suite
[25] http://bbn.com/technology/knowledge/asio_sbrd
[26] http://www.w3.org/Submission/SWRL/
[27] http://d2rq.org/

rules proposed in the W3C's Direct Mapping specification [15].
Mapping implementation: on-demand mapping, data materialisation.
Mapping language features: user-defined URIs, logical table, column selection, column renaming, select condition, vocabulary reuse, 1 table to n classes, many-to-many relation to simple triples, blank nodes, data types, data transformation, user-defined namespaces, static metadata. *NOT supported:* named graphs.
Data retrieval:
- Query-based (SPARQL endpoint).
- Linked Data: allows for dereferencing URIs by providing a representation based on content negotiation: RDF or XHTML. Supports the HTTP 303 dereferencing. Linked Data features: automatically includes rdfs:seeAlso when relevant, XHTML hyperlinks lead to navigation pages containing lists of other resources of the same class, and to an overview page that lists all of these navigation pages. This overview page provides an entry point for external Web search engines to index the content of the database.
The performance varies depending on the access method and is reported to perform reasonably well for basic triple patterns, but there are limitations when SPARQL features such as FILTER, LIMIT are used.
Sustainability: very active project. The last version released in June 2012 supports the W3C's Direct Mapping specification. The first release of DBpedia in 2007 was done using D2R Server, which since then migrated to Virtuoso RDF Views.

4.2.3 *Datalift*

Datalift[28] is an experimental research project which goal is to help users publish and interlink their data sets on the web as Linked Data. Datalift consists in an integrated set of tools that ease the publication process of raw structured data coming from various formats (relational databases, CSV, XML, ...). The tools cover the following functions: selecting ontologies for publishing data, converting data to RDF using the selected ontology, publishing the linked data, interlinking data with other data sets.
Main scope: RDB-to-RDF materialisation-based transformation engine.
Mapping description: The process is done interactively though a web-based GUI: a user selects a data source that is translated following the direct mapping method. Then, a set of modules helps align the RDF data produced on chosen ontologies: RDF-to-RDF transformation (SPARQL CONSTRUCT), renaming of URI (using regular expressions), conversion of strings into URIs, the SILK[29] tool can be used to automatically discover links with other data sets. As a result, although it starts with a Direct Mapping, the different modules provide enough flexibility to perform an a posteriori alignment with capabilities comparable to domain semantics-driven mapping approaches.
Mapping implementation: data materialisation.
Mapping language features: user-defined URIs, column renaming (using the renaming of URIs), data types. Features available through the SPARQL CONSTRUCT module: vocabulary reuse, 1 table to n classes, many-to-many relation to simple triples, blank nodes, named graphs, user-defined namespaces, static metadata. *NOT supported:* logical table, column selection, select conditions, data transformation.
Data retrieval:
- Query-based (SPARQL endpoint).
- Linked Data: allows for dereferencing URIs by providing a representation based on content negotiation: RDF, CSV or XHTML. XHTML hyperlinks lead to navigation pages containing lists of other resources of the same class. Such pages provide an entry point for external Web search engines to index the content of the data sets.
Sustainability: the project will complete in March 2014. A maintenance task force is under discussion.

4.2.4 *DB2OWL*

DB2OWL is a proof of concept that was proposed by Cullot et al. [13]. In the domain of the ontology learning, its goal is to automatically generate a direct mapping describing a relational database, then refine this mapping by exploiting relational schema characteristics: detect many-to-many join tables translated (as simple triples) and concept subsumptions (when a primary key is also a foreign key in another table). The created ontology is expressed in the OWL-DL language, while the mapping description is stored in an R_2O document (see section 4.2.6).
DB2OWL is exclusively focused on the automatic creation of ontology classes and properties that reflect the relational schema; it does not tackle the conversion of the relational data. It must be noticed that most direct mapping implementations now handle the many-to-many relation, whereas this is hardly the case of subsumption relations.
Main scope: ontology learning.
Mapping description: direct mapping augmented with detected many-to-many relations and class subsumption. The mapping description follows the R_2O mapping syntax.
Mapping implementation: n.a. However, in [34], the authors of DB2OWL apply the on-demand mapping implementation to perform the integration of several relational databases using two different levels of ontologies: DB2OWL creates a local ontology for each database and a wrapper converts queries to a local ontology into SQL queries. At the top, user queries expressed on a domain ontology are mapped to queries on local ontologies.
Mapping language features: many-to-many relation to simple triples, column selection. *NOT supported:* user-defined URIs, logical table, select condition, vocabulary reuse, 1 table to n classes, blank nodes, data

[28] http://datalift.org/en
[29] http://silk.semwebcentral.org/

transformation. *Unknown*: column renaming, data types, static metadata, named graphs.
Data retrieval: n.a.
Sustainability: prototype, development inactive since 2007. Source code not available.

4.2.5 *METAmorphoses*

The *METAmorphoses* processor[30] transforms relational data into RDF instances using existing target ontologies. The mappings are specified using a declarative XML-based mapping language [35], and the RDF graph is produced at once (data materialisation) in an RDF/XML output document. METAmorphoses consists in a java library provided under the terms of the LGPL open source license.
Although quite simple, METAmorphoses is an effective tool. It consists in a standalone application that does not require any complex deployment. Mappings can be implemented and tested easily and quickly.
Main scope: generic-purpose mapping language, RDB-to-RDF materialisation-based transformation engine.
Mapping description: domain semantics-driven using a specific declarative XML-based language. The mapping is organised as a two-layer data transformation model. The mapping layer describes how to map database tuples, selected by embedded SQL snippets, to RDF individuals using existing ontological concepts and properties. The template layer describes how to serialize the mapped entities into RDF/XML.
Mapping implementation: data materialisation.
Mapping language features: user-defined URIs, logical tables, column selection, column renaming, select conditions, vocabulary reuse, many-to-many relation to simple triples, 1 table to n classes, user-defined namespaces. *NOT supported:* blank nodes, data types, static metadata, data transformation, named graphs, data transformation.
Data retrieval: none, but the ETL approach can be considered by loading the RDF data materialised into a third party triple store.
Sustainability: no update since 2007.

4.2.6 *R_2O and ODEMapster*

R_2O is a declarative XML-based language [36] that allows the description of complex mappings between existing ontologies and relational elements (relations and attributes). It was initially designed to overcome weaknesses of D2R MAP [33], the predecessor of the current D2RQ mapping language, for cases when the similarity between the relational database and the ontology is low. It provides an extensible set of condition and transformation primitives to address situations such as: 1 table to n classes, joined tables to 1 class, or data transformation. Condition and transformation primitives address features such as string manipulations, arithmetic calculations, definition of order relations, expression of restriction on range of values, etc. Since it was designed, most specific mapping cases addressed by R_2O have been addressed by other projects such as D2RQ, Virtuoso. Besides, when the expressiveness of SQL is not sufficient to deal with those cases, they may be addressed by common database back-end extensions.
R_2O is an evolution of eD2R [37], which addressed mapping situations involving databases that are lightly structured or not in first normal form. eD2R proposed to define complex and conditional transformations on field values based on techniques such as keyword search, regular expression matching, natural language processing and others.
The ODEMapster[31] query engine uses an R_2O document to either execute the transformation in response to a query expressed in the ODEMQL query language, or in a batch mode following the data materialisation approach.
Main scope: generic-purpose mapping language, RDB-to-RDF query-based transformation engine.
Mapping description: domain semantics-driven using a specific declarative XML-based language.
Mapping implementation: ODEMapster supports both on-demand and data materialisation (called massive upgrade) implementations.
Mapping language features: user-defined URIs, logical tables, column selection, column renaming, select conditions, vocabulary reuse, many-to-many relation to simple triples, 1 table to n classes, blank nodes, data types, data transformation. *NOT supported:* named graphs, user-defined namespaces, static metadata.
Data retrieval: query-based. Queries are written in ODEMQL language and stored in an XML file. Query results are retrieved in RDF, as instances of OWL classes. There is no SPARQL result format nor Linked Data. But the ETL approach can be considered by loading the RDF data materialised into a third party triple store.
Sustainability: ODEMapster has been integrated as a plug-in into the NeOn[32] toolkit, an open source ontology engineering environment (ODEMapster plug-in still maintained in last toolkit version Dec. 2011). The EU ADMIRE project reused the NeOn toolkit and provided a distributed access to ODEMapster by using OGSA-DAI [38]. ODEMapster seems to be used only for test purpose within NeOn toolkit. R_2O and ODEMapster will not be maintained in the future as authors are now working on an implementation of R2RML called Morph (see section 4.1.2).

4.2.7 *RDBToOnto*

The RDBToOnto[33] tool was designed in the context of the TAO project[34] (ended beginning of 2009) which

[30] http://www.svihla.net/metamorphoses/METAmorphoses_processor/#documentation

[31] http://neon-toolkit.org/wiki/ODEMapster
[32] http://www.neon-project.org/nw/Welcome_to_the_NeOn_Project
[33] http://www.tao-project.eu/researchanddevelopment/demosanddownloads/RDBToOnto.html
[34] http://www.tao-project.eu/index.html

goal was to allow a fast and effective transition of existing legacy applications to ontologies.
RDBToOnto is a GUI-based extensible tool that enables learning ontologies from relational databases, and it is a framework to ease the development and experimentation of "transitioning methods", i.e. ontology learning from relational databases. To go beyond the automatic generation of a local ontology using the direct mapping method, RDBToOnto proposes a semi-automated method, based not only on the relational database schema but also on the data [39]. The objective is to refine classes derived from the relational schema with subclasses found in the content: find lexical clues in the column names (matched against a predefined list of keywords e.g. "Type"), use data redundancy to discover categorization patterns. Another step allows for database optimization, also known as inclusion dependencies (removal of redundancies using the third party tool LATINO). The whole process is interactive (user defined rules) and iterative. It is a user-oriented tool that also supports the complete transitioning process (ontology learning from a relational database), from the access to the input databases, to the generation of populated ontologies.
RDBToOnto can be extended by means of connectors and converters to implement new learning methods.
Main scope: ontology learning.
Mapping description: basically a direct mapping approach, augmented with semi-automatic iterative process, driven by the database schema and data: lexical clues in the column names, data redundancy suggesting categorization patterns. No mapping language is specified: constraint rules are stored in application specific project files and edited through the GUI only.
Mapping implementation: data materialisation. RDBToOnto generates an ontology optionally populated with instances.
Mapping language features: user-defined URIs, column selection, column renaming, select condition, 1 table to n classes, many-to-many relation to simple triples, data types, user-defined namespaces. *NOT supported:* vocabulary reuse. *Unknown:* logical tables, blank nodes, data transformation, named graphs, static metadata
Data retrieval: none, but the ETL approach can be considered by loading the RDF data materialised into a third party triple store.
Sustainability: RDBToOnto has been used in a large real-world case study in aircraft maintenance[35], including mixing with existing ontologies, maintenance documentation annotation and WSDL annotation. No indication of use and maintenance outside of this context is found.

4.2.8 *Relational.OWL*

The goal of Relational.OWL[36] is to improve the data sharing ability within the community of peer-to-peer databases. Peer-to-peer databases are volatile peers, distributed, their data and schema may evolve frequently. Hence the need for an exchange format that can be understood instantly, that is, without requiring content negotiation nor ontology alignment.
Relational.OWL is an application independent representation technique expressed as an OWL-full ontology, dedicated to the representation of relational schema components by means of a local ontology (e.g. complying with direct mapping rules), as well as the relational data itself. The OWL-full variant of OWL languages is required in order to allow the creation of an ontology which classes are instances of the Relational.OWL ontology classes. The ontology defines classes for tables, columns, primary and foreign keys, data types, and relations (properties) between those classes.
The use of OWL-full to represent the Relational.OWL ontology may be considered as a hurdle to the adoption of the tool, as it is non decidable. Nevertheless authors are "confident of most OWL reasoning tools being able to handle data and schema representations created using Relational.OWL".
Main scope: ontology learning, fully automatic direct mapping.
Mapping description: direct mapping, automatic creation of a local ontology as an instance of the Relational.OWL ontology.
Mapping implementation: data materialisation. The Relational.OWL application is a GUI that automates the process of converting a database into its equivalent OWL full expression.
Data retrieval: none, but the ETL approach can be considered by loading the RDF data materialised into a third party triple store. De Laborda and Conrad [40] propose a method to perform domain-semantics mapping using Relational.OWL: in a nutshell, it consists in applying the ETL approach, then use the SPARQL CONSTRUCT query to perform alignment on a target ontology.
Mapping language features: logical tables, column selection, blank nodes. *NOT supported:* user-defined URIs, column renaming, select conditions, many-to-many relation to simple triples, vocabulary reuse, data transformation, 1 table to n classes, named graphs, static metadata. *Unknown:* data types, user-defined namespaces.
Sustainability: No update since 2006.
Discussion: the point of expressing the schema of a relational database by building an ontology based on the Relational.OWL ontology is interoperability. The fact that two database schemas are described using the same base ontology helps compare classes with classes and properties with properties. However it does not solve alignment issues: for instance, it cannot help figure out that two columns from two databases bear the same semantics. Besides, the same interoperability could be achieved as soon as all mappings use a common method to perform direct mapping like the W3C proposal [15]. De Laborda and Conrad [14] remark that, in other approaches, "the data represented (...) loses its

[35] http://videolectures.net/eswc08_cebrah_tla/
[36] http://sourceforge.net/projects/relational-owl/

relationship to the original database. Tracing the data to its original storage position is thus hardly possible". This is true, but it remains unclear what would be the point of tracing such information. We can think of applications of tracing this information in topics such as the tracing of data provenance, the explanation of query results, or the assessment of confidence in query results.

4.2.9 *SPASQL*

SPASQL[37] is an open-source modified MySQL server, able to parse both SQL and SPARQL queries. The goal of providing MySQL with native support for SPARQL is to allow the same performance as for well-tailored SQL queries, by avoiding the complex rewriting phase [41]: SPARQL and SQL queries are compiled into the same data structures, and are then equally processed by the MySQL query processing engine. SPASQL is an extension of the SQL standard allowing the execution of SPARQL queries within SQL statements, typically by treating them as sub-queries or function clauses. Variables in a SPARQL query are treated like any variable in an SQL query, making the two languages interchangeable to query the same database.
Main scope: RDB-to-RDF query-based transformation engine.
Mapping description: automatic, very basic direct mapping.
Mapping implementation: on-demand. No query rewriting process is needed, the SQL interpreter is extended to support both SQL and SPARQL.
Mapping language features: very limited as only the basic direct mapping applies. *NOT supported:* user-defined URIs, logical tables, column selection, column renaming, select conditions, vocabulary reuse, 1 table to n classes, many-to-many relation to simple triples, blank nodes, data transformation. *Unknown:* user-defined namespaces, named graphs, static metadata, data types.
Data retrieval: SPARQL query-based. Limited support for SPARQL: ASK, CONSTRUCT and DESCRIBE are not supported. SELECT is supported with no variable in predicate position in graph pattern. Results are returned like a usual MySQL result set.
Sustainability: SPASQL has remained in an early prototype status along the project FeDeRate for Drug Research[38]. No update since 2008.

4.2.10 *SquirrelRDF*

With SquirrelRDF[39], Seaborne et al. [42] propose a tool able to apply basic direct mapping rules to an input database or an LDAP directory, and to answer SPARQL queries through an on-demand query rewriting process. Mapping principles roughly follow [12]. The mapping description file (in RDF/Turtle) can be generated automatically from the database, and somewhat customized in order to use an existing target vocabulary. Nevertheless the customization is rather limited, and the mapping essentially remains driven by direct mapping rules. An originality of SquirrelRDF is that is can query multiple databases simultaneously (n tables from m databases).
SquirrelRDF was delivered as part of the Jena framework distribution in 2006, however it is no longer part of it since Jena[40] moved to Apache. A succinct help page is available, but the Javadoc is provided in the SquirrelRDF archive[41].
Main scope: RDB-to-RDF query-based transformation engine using direct mapping.
Mapping description: very simple direct mapping, with no attempt to reveal implicit relations suggested by foreign keys. The mapping description is generated automatically as a Turtle document following a dedicated vocabulary. Nevertheless we could not find a definition of this vocabulary on the internet nor in the downloaded archives.
Besides the basic direct mapping supported, Jena Rules may be used to produce additional knowledge in the perspective of doing more domain-driven mapping.
Mapping implementation: on-demand, SPARQL query-based.
Mapping language features: very limited as only the basic direct mapping applies: column selection, column renaming, vocabulary reuse, blank nodes, static metadata (limited to global metadata, i.e. not attached to each produced RDF triples). *NOT supported:* user-defined URIs, logical tables, select conditions, 1 table to n classes, many-to-many relation to simple triples, data transformation, named graphs. *Unknown:* data types, user-defined namespaces.
Data retrieval: SPARQL query-based. The preferred query mode is the Jena SPARQL API. It can be used to issue SPARQL requests to SquirrelRDF, including SELECT, ASK and CONSTRUCT. A CLI tool is available to test that the configuration is properly set. A limited demo SPARQL endpoint over HTTP supports SELECT. Whatever the access method used, no query triple pattern is allowed with a variable as the property.
Sustainability: no update since 2006.

4.2.11 *Triplify*

The goal of Triplify[42] is to enable popular Web applications (like CMS or blog applications) to publish the content of their relational database in RDF, JSON or Linked Data. Given that many instances of such web applications are deployed over the internet, helping them to quickly expose their database is expected to result in a boost of Semantic Web adoption [43].
Triplify is a simple but nice approach, to be used as a lightweight, easy-to-learn, plug-in for existing web applications. It is based on the mapping of HTTP URI requests onto queries to the relational database. Compared to the direct mapping approach, Triplify

[37] http://www.w3.org/wiki/SPASQL
[38] http://www.w3.org/2004/10/04-pharmaFederate/
[39] http://jena.sourceforge.net/SquirrelRDF/
[40] http://jena.apache.org/download/index.html
[41] http://sourceforge.net/projects/jena/files/SquirrelRDF/
[42] http://triplify.org/

takes the problem the other way round: it first focuses on the relational data that actually matters, instead of translating the relational schema into an ad-hoc ontology and then populates this ontology. Mappings are implemented as SQL statements embedded in PHP scripts, therefore any SQL construct or aggregation function may be used. Authors argue that "Triplify facilitates the creation of custom-tailored search engines targeted at certain niches, e.g. searching for specific content in various blogs, wikis, or forums".
It must also be mentioned that some Triplify modules specifically address the provenance metadata generation (using the Provenance Vocabulary[43]) as well as the publication of update logs.
Main scope: generic-purpose mapping language, RDB-to-RDF transformation engine (query-based and transformation-based).
Mapping description: domain semantics-driven manual mapping by means of SQL queries embedded in PHP scripts, in addition to transformation functions written in PHP. May use any SQL construct and aggregation function.
Mapping implementation: on demand (Linked Data), data materialisation.
Mapping language features: user-defined URIs, logical tables, column selection, column renaming, select conditions, vocabulary reuse, 1 table to n classes, many-to-many relation to simple triples, blank nodes, data types, data transformation, user-defined namespaces, static metadata. *NOT supported:* named graphs.
Triplify also naturally supports the *Link Data Update Logs*, by interpreting URIs that contain the update keyword, and mapping them to appropriate queries for changes in the database.
Data retrieval: SPARQL is not supported. Data is retrieved by URI as Linked Data (RDF, JSON). Authors state that Triplify is aimed at small to medium Web applications (i.e. less than 100MB database content). However, it supports the caching of the triplification results and can hence be used with large Web applications (160BG data for OpenStreetMap). It has been adapted to several popular web applications (WordPress, Joomla, osCommerce, etc.).
Sustainability: Provided under the terms of the LGPL licence. The last version dates back to March 2010.

4.3 Summary

Table 2 summarizes the characteristics of all the tools studied in this section.

[43] http://sourceforge.net/apps/mediawiki/trdf/index.php?title=Provenance_Vocabulary

	Tool	Motivation			Mapping description		Mapping implementation		Data retrieval			Features of mapping description languages														
		Ontology learning	Generic-purpose mapping language	Transformation engine	Direct mapping	Domain Semantics-Driven mapping	Data materialisation	On-demand mapping	Query-based access	Linked data	Graph dump	User defined unique Ids	Logical table	Column selection	Column renaming	Select conditions	Vocabulary reuse	1 table to n classes	many-to-many to simple triples	Blank nodes	Data types	Data transformation	Named graphs	User-defined namespaces	Static metadata
Non R2RML	D2RQ		X	X		X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X		X	X
	Datalift			X	X	X	X		X	X		X			X		X	X	X	X	X		X	X	X
	DB2OWL	X			X		n.a.		n.a.					X	?				X		?			?	?
	METAmorphoses		X	X		X	X		n.a.				X	X	X	X	X	X	X		X			X	X
	R_2O/ODEMapster		X	X		X	X	X	X			X	X	X	X	X	X	X	X	X	X	X		X	
	RDBToOnto	X			X		X		n.a.			X	?	X	X	X		X	X	?	X	?	?	X	?
	Relational.OWL	X			X		X		n.a.				X	X						X	?			?	
	SPASQL			X	X			X	X												?		?	?	?
	SquirrelRDF			X	X			X	X					X	X		X			X	?			?	?
	Triplify		X	X		X	X	X		X		X	X	X	X	X	X	X	X	X	?	X		X	X
R2RML	DB2Triples			X		X		X	X			X	X	X	X	X	X	X	X	X	X		X	X	X
	Morph			X		X	X	X	X			X	X	X	X	X	X	X	X		X	?	X	X	
	Oracle 12c			X		X		X	X			?	?	?	?	?	?	?	?	?	?	?	?	?	?
	RDF-RDB2RDF			X		X	X		n.a.			X	X	X	X	X	X	X	?	X	?		X	X	X
	Virtuoso RDF Views		X	X		X		X	X	X		X	X	X	X	X	X	X	X	X	?	?	?	X	?
	Ultrawrap			X		X		X	X	X		X	X	X	X	X	X	X	X	X	X	?	X	X	X
	XSPARQL		X	X		X	X		X			X		X	X	X	X	X	X	X	X	X		X	X

Table 2 - Synthetic view of tools features

Commercial product Asio SBRD is not listed as too few technical information is available publicly.
The "X" cross means supported, an empty cell means unsupported. Interrogation marks mean that available information did not permit to figure out if the feature is supported or not. In such cases, experimentations would be required to conclude.

5 SYNTHESIS

The previous section shows a large diversity of approaches among the tools designed to translate relational databases into RDF. Overall, the various approaches are driven by different initial motivations that fall in one or several of: ontology learning from a relational database, definition of a generic-purpose mapping language and implementation of an RDB-to-RDF transformation engine. This has been presented in section 2.1. Whatever the motivation though, three major steps cannot be circumvented: the mapping description (direct vs. domain semantics-driven), the mapping implementation (RDF data materialisation or on-demand) and the data retrieval (query-based access, linked data or graph dump). These steps have been described in sections 2.2, 2.3 and 2.4.
In this section, we analyse the studied tools with regards to the criteria identified in section 2. We first provide some conclusions regarding the motivations of the tools studied, then we examine how the tools orchestrate the three mapping steps.

5.1 Motivations

Table 2 evidences several patterns with regards to the motivation of the tools studied, that we describe below.

Ontology learning uses direct mapping
The three ontology learning approaches studied in this paper (DB2OWL, RDBToOnto, Relational.OWL) apply the direct mapping description (Table 2, columns *Ontology learning* and *Direct Mapping*). This indeed appears as a natural choice of ontology learning tools, since the goal is not to align the database on existing ontologies (that would require a domain semantics-driven mapping) but to discover ontological knowledge from the database. Sequeda et al. [10] clearly illustrate this close connection between ontology learning and direct mapping in their review of Direct Mapping approaches: all of the seven papers reviewed mainly address the automatic ontology learning through the use of direct mapping principles. Nevertheless not all ontology learning approaches are based on the automatic direct mapping, as illustrated by RDBToOnto in which the direct mapping is used to produce raw RDF data from which more elaborated concepts and properties are deducted along an interactive and iterative process.

Ontology learning uses data materialisation
Two of the ontology learning approaches studied (RDBToOnto and Relational.OWL) apply the data materialisation implementation, while DB2OWL does not provide any implementation. By contrast with the direct mapping, the data materialisation implementation does not appear as an obvious choice. It is most likely the result of a pragmatic choice, that is, such approaches need to demonstrate the method without implementing a complex on-demand translation engine. Hence, they make the choice of the data materialisation method, which implementation is more straight-forward. Again, Sequeda et al. [10] confirm this observation as all of the seven papers that they reviewed apply the data materialisation implementation method.

Generic-purpose mapping language vs. transformation engine
When the motivation is the definition of a generic-purpose mapping language, it is accompanied by a transformation engine. The engine implements the mapping language defined, at least to demonstrate the capabilities of the language, and possibly to be a production-class solution integrating the mapping language and the transformation engine. Obviously, R2RML is the only exception to this rule (it does not go along with an implementation), since its goal is, on the contrary, to serve as a standard for the implementation of transformation engines.
The definition of a generic-purpose mapping language always goes along with a domain semantics-driven mapping description. The reason is quite simple: since the direct mapping can be automated, there is no point to defining a mapping language which expressiveness would solely support the direct mapping. Therefore the definition of a mapping language always targets a domain semantics-driven mapping.

5.2 A constellation of RDB-to-RDF approaches

Figure 1 presents the whole RDB-to-RDF process which results of the sequential combination of the three mapping steps: the mapping description, the mapping implementation and the data retrieval. It also depicts the different options applicable at each step, and the way those options can be chained to achieve the whole process. The figure reads from top to bottom, following the chronological order involved when setting up the translation process. Conversely, read from bottom to top, the figure illustrates the way the RDF data is retrieved, how this retrieval method is handled based on the mappings implementation, up to the source relational data. Arrows denote possible transitions. Blue arrows denote paths starting with a direct mapping description, while red arrows denote paths starting with a domain semantics-driven mapping description. Dotted arrows denote transitions that are technically possible but not common in practice.

5.2.1 *Composition of mapping descriptions and mapping implementations*

With the insight gained from the different approaches studied in this paper, it turns out that the mapping description method (top layer in Figure 1) discriminates approaches into two families with regards to the overall RDB-to-RDF process.
(i) The Direct Mapping family gathers either light approaches which goal is to quickly and simply expose relational data to the world (SquirrelRDF, SPASQL), and ontology learning approaches which aim at making

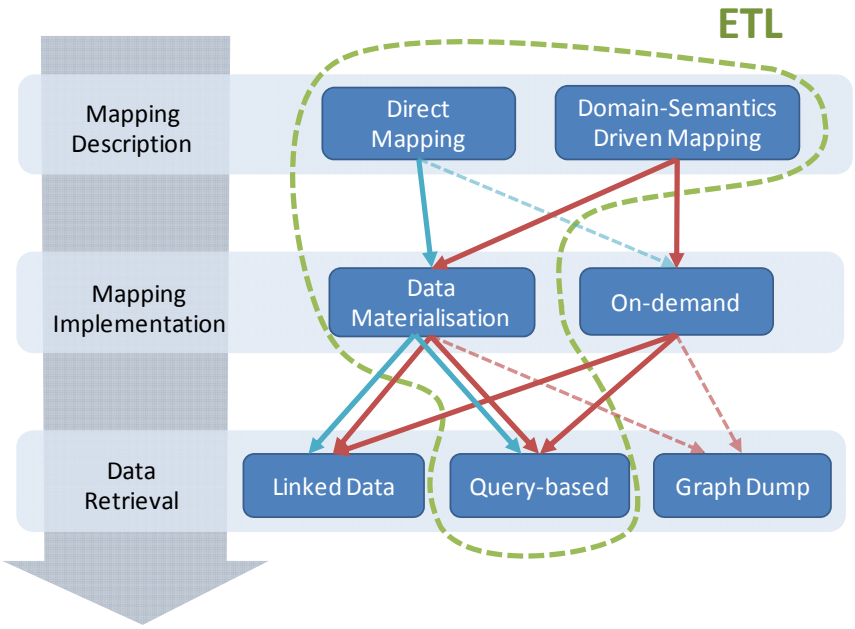

Figure 1: Steps of the RDB-to-RDF process
From top to bottom, the figure depicts the steps needed to set up the translation process. Blue paths start with a direct mapping description, red paths start with a domain semantics-driven mapping description. Dotted arrows denote unusual sequences.

explicit the semantics of relational databases (DB2OWL, RDBToOnto, Relational.OWL). In the first case, an RDB-to-RDF transformation engine is needed to make the data available; SquirrelRDF and SPASQL both use the on-demand mapping implementation. Conversely, in the case of ontology learning approaches, the transformation engine is optional but generally comes from the need to implement a proof of concept. RDBToOnto and Relational.OWL apply the data materialisation principle, whereas DB2OWL does not provide any transformation engine at all.

(ii) In the Domain Semantics-Driven Mapping family, tools are more generally focused on the definition of an expressive generic-purpose mapping language (D2RQ, METAmorphoses, R_2O/ODEMapster, Triplify, Virtuoso RDF Views). To some varying extend, mapping languages rely on a mix of specific constructors and embedded SQL snippets. Triplify even solely relies on SQL. Recent tools do not define a new mapping language but comply with R2RML (DB2Triples, Morph, Oracle 12c, RDF-RDB2RDF, Ultrawrap), or complement an existing mapping language with the support of R2RML (XSPARQL, Virtuoso RDF Views).

In addition, tools of the Domain Semantics-Driven Mapping family generally entail the implementation of an efficient RDB-to-RDF transformation engine. This is even a major motivation of R2RML-compliant tools, as attested by Table 2 (R2RML tools, column "Transformation engine"). Virtuoso and XSPARQL are a specific case as they come with their own mapping description language in addition to supporting R2RML. The mapping implementation can be either on-demand (Asio SBRD, D2R Server, R_2O/ODEMapster, Triplify, DB2Triples, Oracle 12c, Virtuoso RDF Views, Ultrawrap) or based on the data materialisation principle (DataLift, METAmorphoses, Morph, RDF-RDB2RDF, XSPARQL).

Direct Mapping and Data Materialization

The direct mapping approach is generally implemented using the data materialisation technique. Indeed, given that the direct mapping produces semantically poor RDF data, exposing this data as is with an on-demand implementation (rewriting of queries at run time) may not be of much interest. Instead, materialising this data offers the opportunity to align it on existing ontologies, or to infer new knowledge using a rule engine or a reasoner, and then to load it into a query engine to expose the semantically enriched data. This is in particular the case of ontology-learning approaches.

The direct mapping method may also be implemented with the on-demand method, denoted by the blue dotted arrow in the upper half of Figure 1. This is hardly the case in the tools we have studied: only SquirrelRDF and SPASQL do so, but both remained in a prototype stage and are now deprecated (note that we consider here only strictly direct mapping tools, as opposed to the domain semantics-driven tools that provide a direct mapping tool for convenience). Nevertheless, this may only be a bias of the set of tools studies, and there is no reason why this combination (direct mapping and on-demand implementation) should not be used in other tools. As an illustration, here is a hypothetical scenario: let us assume a versatile environment where databases are volatile and may appear only once, or where their schema may evolve frequently (situation addressed by Relational.OWL). Those databases are exposed automatically using the direct mapping method and the

on-demand implementation through a SPARQL endpoint. Provided with a way to discover such databases, a federation system may query the local ontology of each database and use automatic ontology alignment methods (e.g. natural language processing or syntactical coincidences such as those investigated by RDBToOnto) to perform a best-effort alignment of the different databases on each other or on a global ontology.

5.2.2 *Data Retrieval methods*

The Query-based retrieval method is the most implemented within the tools we have studied, as evidenced by the "Query-based access" column in Table 2. Triplify supports exclusively the Linked Data method, whereas all other tools supporting Linked Data also support the Query-based method.

By contrast, the red dotted arrows, in the lower half of Figure 1, show that the graph dump data access (get a representation of the full graph using an HTTP GET request) is very unusual, for several reasons. Firstly, the data materialisation process produces a graph serialization, generally in the form of an RDF file. Hence, using the graph dump access method would be just another way of getting an alternative representation of the same graph from an HTTP endpoint. Secondly, in conjunction with the on-demand mapping implementation, the dump of large graphs may induce significant performance issues. Thus, so far, the graph dump seems not to have been a priority for RDB-to-RDF tools developers (none of the studied tools supports it, as evidenced by the empty column in Table 2). Nevertheless, this may become more common in the future, as more tools will implement full SPARQL 1.1 compliant endpoints, i.e. including the SPARQL 1.1 Graph Store Protocol.

5.2.3 *Extract-Transform-Load (ETL)*

The Extract-Transform-Load (ETL) expression is used in the literature to denote the approach that consists in defining a mapping, materialising the RDF data based on that mapping, and loading the RDF data into a triple store to query it, typically through a SPARQL endpoint. The green dotted line in Figure 1 denotes this whole approach. It shows that any of the direct or domain semantics-driven mappings can apply. Strictly speaking, and as there is no formal definition of the ETL approach, the Linked Data and Graph Dump data retrieval methods could also be included. In practice though, ETL generally refers to the query-based retrieval method, and more precisely the usage of a SPARQL endpoint.

5.2.4 *A posteriori production of rich semantic data*

Producing RDF data with sufficient semantics, in order to make it usable, interoperable and linkable, is a critical concern in most RDB-to-RDF tools. Experimentations that use direct mapping-based tools often underline the weaknesses of this method: the reuse of the RDF data produced is not easy due to the poorness of the semantic reference, its interoperability with other data sets requires complex techniques. Finally, lifting the data to a higher level of semantic formalisation, in other words, aligning the RDF data with existing ontologies has to be done at some point anyway. Furthermore, even in the case of a domain semantics-driven mapping, the production of sufficiently rich semantic data cannot always be achieved by the expressiveness of existing mapping languages. Consequently, various strategies are investigated in the literature to produce additional semantic data, a posteriori. Below we briefly present some of them.

Aligning multiple levels of ontology

Several RDB-to-RDF projects propose to enhance the direct mapping by aligning the local ontology with higher levels of abstraction formalized by domain and application ontologies. In this case, the domain semantics-mapping is not performed beforehand as in domain semantics-driven mapping tools, but consists in the later alignment of ontologies.

Asio SBRD, DataLift, as well as a DB2OWL-based approach [34], propose comparable methods using two distinct levels of ontology. In a first step, the direct mapping method is applied to create a local ontology that reflects the structure of the relational database. In a second step, the local ontology is manually aligned onto a domain ontology that models some part of the domain which the relational database refers to. A query engine translates a query to the domain ontology into a query to the local ontology. In turn, the query to the local ontology is translated into an SQL query to the relational database.

Alternatively, the alignment of the local ontology with a domain ontology may be achieved using the SPARQL CONSTRUCT clause: given a certain graph pattern using local ontology concepts and properties, the CONSTRUCT clause produces new RDF data using concepts and properties of the domain ontology. De Laborda and Conrad [40] describe a use case in which they have applied this method with Relational.OWL. Technically, the SPARQL CONSTRUCT clause returns a graph instance, it does not insert triples into an existing graph. Consequently, the alignment has to be done unitarily in a single CONSTRUCT clause. Anyway, it would be simple to apply the same idea by running several SPARQL INSERT clauses to incrementally enrich a graph. Ultimately, this method turns out to be equivalent to executing rules in a rule engine (see below). The only difference, here, is that the rules description language would be SPARQL.

Hu and Qu [44] propose to apply data mining techniques to automatically discover simple mappings between the relational entities (tables, columns, integrity constraints) and the classes and properties of an existing ontology. Relational and ontological entities are compared using distance measures such as the TF-

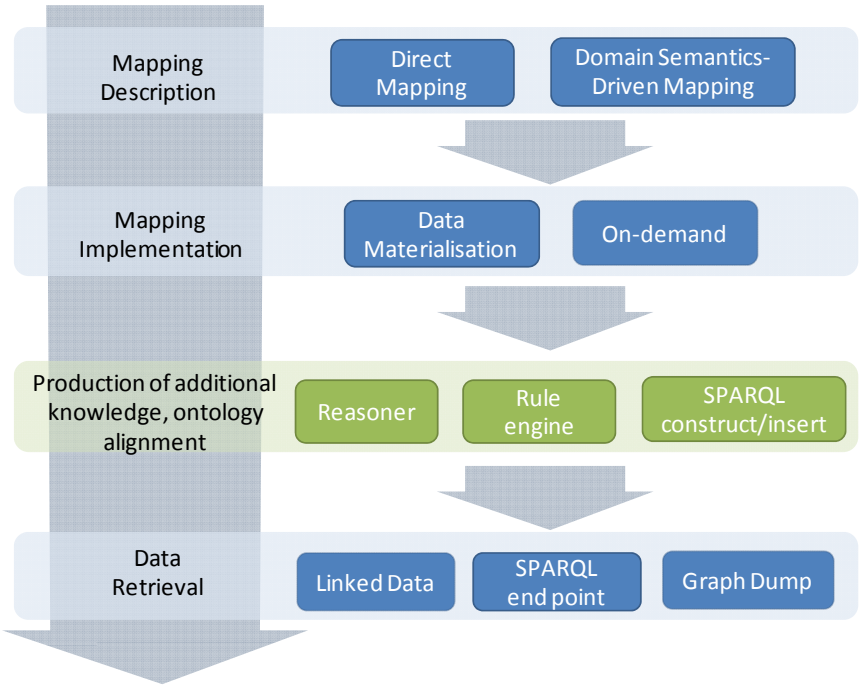

Figure 2: Production of additional knowledge
The green layer depicts strategies aimed at a posteriori production of additional semantic data

IDF (*Term Frequency-Inverse Document Frequency*), in addition to the computation of the confidence in a mapping and a validation phase. In this sense, it can be seen as an automatic domain semantics-driven approach. However, for this method to perform efficiently, the similarity between the relational schema and the ontology should be high, unless human users provide initial reference mappings.

Besides generic methods such as presented above, some domain-specific projects propose alternative methods. Green et al. [45] describe a demonstrator that relies on existing tools to allow for the "spatial attribution" of data sources, for the predictive modelling of diffuse water pollution. The federation of different data sources lies on a 3-layers ontology stack. The ontologies at the first level (called data ontologies) are used to map each data source to concepts of the ontologies at the next level (called domain ontologies). Domain ontologies are written by domain experts, each data ontology has a single corresponding domain ontology. At the top, the application ontology links the domain ontologies together with the addition of application-specific information. Much of the application ontology is created manually, as it requires domain knowledge of the scenario being modelled. D2RQ is used to map data sources to the data ontologies, and is extended to include spatial operators provided by the Oracle Spatial database. In particular, this extension helps decide on which domain concept to map data based on the result of specific functions such as the calculation of a perimeter. The virtual RDF graph thus generated from the data sources can be searched by means of SPARQL queries, those queries can use concepts and properties of the application and domain ontologies. Additionally, the Pellet reasoner is used to enrich results by benefitting of the full semantics of OWL.

Using rules to create additional knowledge

Rule engines or reasoners are an alternative way to enrich the semantics of RDF data. This option is depicted on Figure 2 by the green layer inserted between the mapping implementation and the data retrieval layers.

Techniques proposed in this layer can easily be used in conjunction with the data materialisation in which the whole graph is available at once. This is the case of the NeuroLOG project [46] that uses METAmorphoses to create an RDF materialisation of federated neuro-imaging relational databases. The values of some columns help deduce the belonging to classes of the application ontology. As the METAmorphes mapping language is not capable of describing such categorization patterns, these patterns are described as SWRL[44] rules executed by the CORESE [47] semantic search engine. By raising the quality of the semantic data, the process improves the possibilities of further application specific reasoning.

Rule engines or reasoners can also apply on RDF data translated on-demand, although the execution of complex deduction rules, or the application of complex entailments, may generate a high volume of queries to the database. In the worst case, the process may require to translate the whole relational data before being able to complete. Among the tools studied in this paper, several propose the support of rules: SWRL rules for Asio, Silk module in DataLift, Jena Rules in

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

SquirrelRDF, and proprietary RDF-to-RDF deduction rules in XSPARQL.

6 Open Questions

The RDB-to-RDF methods that we have described in this document are one of the keys to populate the web of data, by publishing in RDF the immense amount of data currently locked in relational databases. More generally, many initiatives target the publication of non-RDF data sources in RDF, ranging from wrappers of popular web site APIs, such as the *flickrdf*[45] library, to miscellaneous format-specific converters[46] and "RDFizers"[47]. Whatever the method though, all of this results in a read-only web of data. And similarly to the tremendous change of paradigm that lead to the Web 2.0 where the user contribution plays a major role, the web of data will necessarily have to shift to a write-enabled model. In the case of relational databases, although the expressiveness of RDB-to-RDF mappings is a key to rich semantic alignment, it also often results in one-way transformation mappings (one-way mappings cannot be used to revert RDF data to the original relational data). Consequently, there will necessarily be a compromise to be made between the expressiveness of RDB-to-RDF mapping languages and the need for updating relational data using the protocols of the semantic web. The R3M mapping language is an example of the search for such a compromise [48].
Berners-Lee et al. [49] investigate some of the issues and challenges with regards to a read-write web of data. They underline that the need to create, update and delete RDF data should be made possible in a secure, reliable, trustworthy and scalable way. This questions the way to authenticate people and programs in the web of data, how to securely assign them authorizations, how to track the changes made, etc. As to the trustworthiness question, in a global data space made of heterogeneous data sources integrated together, it will become critical to be able to explain query results, or to assess the confidence a user can have in query results. To this end, applications exposing non-RDF data sources into RDF will have to ensure the preservation of provenance information, allowing to track a piece of data back to its original data source.

7 Conclusion

Many techniques and tools have been proposed over the last years to enable the publication of relational data on the web in RDF. In this paper, we have described four main axes along which it is possible to categorize such RDB-to-RDF approaches. Based on these axes, we have proposed a detailed review of seventeen RDB-to-RDF tools, either supporting the R2RML W3C recommendation or providing their own mapping language. The categorization proposed helped us identify commonalities and differences between existing approaches, as well as the correlations between the solutions applicable to each step of the RDB-to-RDF translation process. We observed that producing RDF data with sufficiently rich semantics is a critical concern of most approaches studied, in order to make the data usable, interoperable and linkable. Therefore, we have briefly presented various strategies investigated in the literature to produce richer semantic data.

[45] http://librdf.org/flickcurl/
[46] http://www.w3.org/wiki/ConverterToRdf
[47] http://simile.mit.edu/wiki/RDFizers

8 References

[1] B. He, M. Patel, Z. Zhang, K.C.-C. Chang, Accessing the deep web, Communications of the ACM. 50 (2007) 94–101.

[2] T. Berners-Lee, Linked Data, in Design Issues of the WWW, (2006).

[3] A. Ruttenberg, T. Clark, W. Bug, M. Samwald, O. Bodenreider, H. Chen, et al., Advancing translational research with the Semantic Web, BMC Bioinformatics. 8 (2007) S2.

[4] A. Burgun, O. Bodenreider, others, Accessing and integrating data and knowledge for biomedical research, Yearb Med Inform. 47 Suppl. 1 (2008) 91–101.

[5] H. Akil, M.E. Martone, D.C. Van Essen, Challenges and Opportunities in Mining Neuroscience Data, Science. 331 (2011) 708–712.

[6] S. Sahoo, W. Halb, S. Hellman, K. Idehen, T. Thibodeau, S. Auer, et al., A Survey of Current Approaches for Mapping of Relational Databases to RDF, (2009).

[7] M. Hert, G. Reif, H.C. Gall, A comparison of RDB-to-RDF mapping languages, in: Proceedings of the 7th International Conference on Semantic Systems, ACM, Graz, Austria, 2011: pp. 25–32.

[8] S. Auer, L. Feigenbaum, D. Miranker, A. Fogarolli, J. Sequeda, Use Cases and Requirements for Mapping Relational Databases to RDF, (2010).

[9] J. Sequeda, Relational Database and the Semantic Web, (2010).

[10] J. Sequeda, S.H. Tirmizi, Ó. Corcho, D.P. Miranker, Survey of directly mapping SQL databases to the Semantic Web., Knowledge Eng. Review. 26 (2011) 445–486.

[11] S. Tirmizi, J. Sequeda, D. Miranker, Translating sql applications to the semantic web, in: Proceedings of the 19th International Conference on Database and Expert Systems Applications (DEXA'08), 2008: pp. 450–464.

[12] T. Berners-Lee, Relational Databases and the Semantic Web, in Design Issues of the WWW, (1998).

[13] N. Cullot, R. Ghawi, K. Yetongnon, DB2OWL : A Tool for Automatic Database-to-Ontology Mapping, in: Proceedings of the 15th Italian Symposium on Advanced Database Systems

(SEBD 2007), Torre Canne, Fasano, BR, Italy, 2007: pp. 491–494.
[14] C.P. De Laborda, S. Conrad, Relational.OWL: a data and schema representation format based on OWL, in: Proceedings of the 2nd Asia-Pacific Conference on Conceptual Modelling, Newcastle, Australia, 2005: pp. 89–96.
[15] W3C RDB2RDF WG, A Direct Mapping of Relational Data to RDF, (2012).
[16] C. Dolbear, J. Goodwin, Position paper on Expressing Relational Data as RDF, in: Proc. of W3C Workshop on RDF Access to Relational Databases, 2007: pp. 25–26.
[17] W3C, SPARQL 1.1 Query Language, (2013).
[18] O. Erling, Requirements for Relational to RDF Mapping, (2008).
[19] O. Erling, Requirements for Relational-to-RDF Mapping, blog additional, Orri Erling's Weblog. (2008).
[20] W3C, SPARQL 1.1 Protocol, (2013).
[21] W3C TAG, Dereferencing HTTP URIs - Draft Tag Finding 31 August 2007, (2007).
[22] W3C, SPARQL 1.1 Graph Store HTTP Protocol, (2013).
[23] W3C RDB2RDF WG, R2RML: RDB to RDF Mapping Language, (2012).
[24] M. Dean, Suggestions for Semantic Web Interfaces to Relational Databases, in: W3C Workshop on RDF Access to Relational Databases, Cambridge, MA, USA, 2007.
[25] W3C RDB2RDF XG, W3C RDB2RDF Incubator Group Report, (2009).
[26] Oracle, Oracle Spatial and Graph 12c, RDF Semantic Graph, (2013).
[27] Oracle, Semantic Technologies in Oracle Database 11g Release 2: Capabilities, Interfaces, Performance, (2010).
[28] J.F. Sequeda, R. Depena, D.P. Miranker, Ultrawrap: Using sql views for rdb2rdf, in: Proceedings of the 8th International Semantic Web Conference (ISWC2009), 2009.
[29] N. Lopes, S. Bischof, S. Decker, A. Polleres, On the Semantics of Heterogeneous Querying of Relational, XML and RDF Data with XSPARQL, in: Proceedings of the 15th Portuguese Conference on Artificial Intelligence (EPIA2011), Lisbon, Portugal, 2011.
[30] C. Bizer, R. Cyganiak, D2R server - Publishing relational databases on the semantic web, in: 5th International Semantic Web Conference, 2006: p. 26.
[31] C. Bizer, A. Seaborne, D2RQ-treating non-RDF databases as virtual RDF graphs, in: Proceedings of the 3rd International Semantic Web Conference (ISWC2004), 2004: p. 26.
[32] R. Cyganiak, C. Bizer, O. Maresch, C. Becker, The D2RQ Mapping Language v0.8, (2012).
[33] C. Bizer, D2R MAP - A Database to RDF Mapping Language, in: Proceedings of the 12th International World Wide Web Conference, Budapest, Hungary, 2003.
[34] R. Ghawi, N. Cullot, Database-to-ontology mapping generation for semantic interoperability, in: Third International Workshop on Database Interoperability (InterDB 2007), Held in Conjunction with VLDB 2007, Vienna, Austria, 2007.
[35] M. Svihla, I. Jelinek, Two layer mapping from database to RDF, in: Proceedings of Electronic Computers and Informatics (ECI), 2004.
[36] J. Barrasa, Ó. Corcho, A. Gómez-Pérez, R2O, an extensible and semantically based database-to-ontology mapping language, in: Proceedings of the Second Workshop on Semantic Web and Databases (SWDB 2004), Springer-Verlag, Toronto, Canada, 2004.
[37] J. Barrasa, O. Corcho, A. Gómez-Pérez, Fund Finder: A case study of database-to-ontology mapping, in: Semantic Integration Workshop, in Proceedings of the 2nd International Semantic Web Conference (ISWC-2003), Sanibel Island (FL, US), 2003.
[38] F. Priyatna, RDF-based access to multiple relational data sources, Universidad Politécnica de Madrid, 2009.
[39] F. Cerbah, Learning highly structured semantic repositories from relational databases: The RDBToOnto tool, in: Proceedings of the 5th European Semantic Web Conference on The Semantic Web (ESWC 2008), Athens, GA, USA, 2008: pp. 777–781.
[40] C.P. De Laborda, S. Conrad, Database to Semantic Web Mapping using RDF query languages, Conceptual Modeling-ER 2006. (2006) 241–254.
[41] E. Prud'hommeaux, SPASQL: SPARQL Support In MySQL, (2007).
[42] A. Seaborne, D. Steer, S. Williams, RDF-SQL (SquirrelRDF), (2007).
[43] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, D. Aumueller, Triplify: light-weight linked data publication from relational databases, in: Proceedings of the 18th International Conference on World Wide Web, Madrid, Spain, 2009: pp. 621–630.
[44] W. Hu, Y. Qu, Discovering simple mappings between relational database schemas and ontologies, The Semantic Web. (2007) 225–238.
[45] J. Green, C. Dolbear, G. Hart, J. Goodwin, P. Engelbrecht, Creating a semantic integration system using spatial data, in: Proceedings of the 7th International Semantic Web Conference (ISWC2008), Karlshue, Germany, 2008: pp. 26–30.
[46] B. Gibaud, G. Kassel, M. Dojat, B. Batrancourt, F. Michel, A. Gaignard, et al., NeuroLOG: sharing neuroimaging data using an ontology-based federated approach, in: AMIA Annual Symposium Proceedings, 2011: p. 472.

[47] O. Corby, R. Dieng-Kuntz, C. Faron-Zucker, Querying the semantic web with corese search engine, in: Proceedings of 16th European Conference on Artificial Intelligence, Valencia, Spain, 2004: p. 705.
[48] M. Hert, G. Reif, H.C. Gall, Updating relational data via SPARQL/update, in: Proceedings of the 2010 EDBT/ICDT Workshops, 2010: p. 24.
[49] T. Berners-Lee, R. Cyganiak, M. Hausenblas, J. Presbrey, O. Seneviratne, O.-E. Ureche, Realising A Read-Write Web of Data, 2009.