Machine-Interpretable Dataset and Service Descriptions for Heterogeneous Data Access and Retrieval

Anastasia Dimou
anastasia.dimou@ugent.be

Ruben Verborgh
ruben.verborgh@ugent.be

Miel Vander Sande
miel.vandersande@ugent.be

Erik Mannens
erik.mannens@ugent.be

Rik Van de Walle
rik.vandewalle@ugent.be

Ghent University – iMinds – Multimedia Lab
Ghent, Belgium

ABSTRACT

The RDF data model allows the description of domain-level knowledge that is understandable by both humans and machines. RDF data can be derived from different source formats and diverse access points, ranging from databases or files in CSV format to data retrieved from Web APIs in JSON, Web Services in XML or any other speciality formats. To this end, machine-interpretable mapping languages, such as RML, were introduced to uniformly define how data in multiple heterogeneous sources is mapped to the RDF data model, independently of their original format. However, the way in which this data is accessed and retrieved still remains hard-coded, as corresponding descriptions are often not available or not taken into account. In this paper, we introduce an approach that takes advantage of widely-accepted vocabularies, originally used to advertise services or datasets, such as Hydra or DCAT, to define how to access Web-based or other data sources. Consequently, the generation of RDF representations is facilitated and further automated, while the machine-interpretable descriptions of the connectivity to the original data remain independent and interoperable, offering a granular solution for accessing and mapping data.

Keywords

Linked Data Mapping, Data Access, Data Retrieval, RML

1. INTRODUCTION

Describing domain-level knowledge, understandable both by humans and machines, can be achieved by representing data using the RDF data model. Although, obtaining its semantic representation requires dealing with data which can originally (i) reside on *diverse, distributed locations*, (ii) be approached using *different access interfaces* and (iii) have *heterogeneous structures and formats*:

Diverse, distributed locations
: Data can reside locally, e.g., in files or in a database at the local network, or can be published on the Web.

Different access interfaces
: Data can be approached using diverse interfaces. For instance, it can be as straightforward to access the data as raw files. There might be metadata that describe how to access the data, as in the case of data catalogues. However, it might also be required to have a dedicated access interface to retrieve the data from a repository, such as database connectivity for databases, or different interfaces from the Web, such as Web APIs.

Heterogeneous structures and formats
: Data can be stored and/or retrieved in different structures and formats. For instance, data can originally have a *tabular structure*, (e.g., databases or CSV files), be *tree-structured* (e.g., XML or JSON format), or be *semi-structured* (e.g., in HTML).

Incorporating ever-increasing amounts of data from multiple sources in different formats into a common knowledge domain is challenging. Despite the significant number of existing tools (Section 3), integration remains complicated. More precisely, most tools that generate RDF representations map from a single source format and a given input to RDF. Only few provide mappings from different source formats to RDF, and even fewer provide independent, interoperable and machine-interpretable mapping definitions.

Even though uniform definitions for how to map heterogeneous data to the RDF data model have been addressed [10], more generic application still cannot be built because data access and retrieval remains hard-coded. In particular, uniform, machine-interpretable mapping definitions indicate how triples should be generated in a generic way for all possible different input sources. Those mapping definitions contain references to an input data source, which are case-specific and, thus, defined using formulations relevant to the corresponding data format, e.g., XPath for data in XML format. However, as data access and retrieval remains out of the scope of mapping definitions, it ends up being hard-coded in the corresponding implementations. While this is not a major problem when local, custom, or input-specific data is considered, the situation aggravates when data from multiple heterogeneous data sources, accessed via different interfaces, is required to be retrieved and mapped to the RDF data model.

SEMANTiCS '15, September 15 - 17, 2015, Vienna, Austria

ISBN 978-1-4503-3462-4/15/09...$15.00

DOI: http://dx.doi.org/10.1145/2814864.2814873

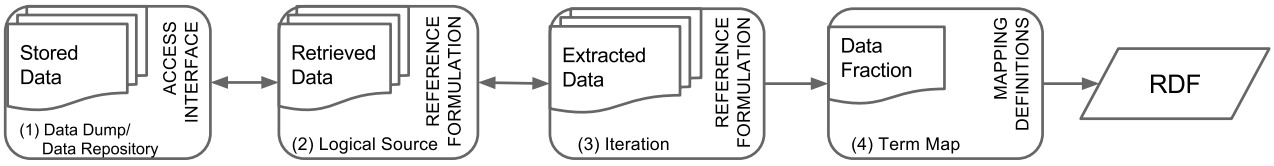

Figure 1: Data retrieval and mapping to the RDF data model.
A triple consists of RDF Terms which are generated by Term Maps (4). Those Term Maps are defined with a *mapping definition vocabulary* and are instantiated with data fractions referred to using a *reference formulation* relevant to the corresponding data format. Those fractions are derived from data extracted at a certain iteration (3) from a logical source (2). Such a logical source is formed by data retrieved from a repository (1) which is accessed as defined using the corresponding *dataset or service description vocabulary*.

Vocabularies which are originally used to advertise datasets or services (e.g., DCAT[1] or Hydra[2]) and to enable applications to easily consume the underlying data exist. These same vocabularies can be used to specify how to access and, subsequently, retrieve data, available on the Web or not and generate their RDF representation. This way, the description that specifies how to access the data becomes machine-interpretable, as the mapping descriptions are. However, access descriptions with such vocabularies are not taken into account and are not aligned with the vocabularies used to describe the mapping definitions so far.

In this paper, we introduce an approach that exploits W3C-recommended or widely-accepted vocabularies, originally used to advertise datasets or services, e.g., SPARQL-SD[3] or DCAT, to define how to access data sources, available on the Web or not, and generate their RDF representation. Our contribution is twofold: (i) we review different vocabularies of interfaces that describe how to access data and we specify how data sources can be instantiated using those descriptions; (ii) we define how such access interface descriptions can be aligned with a mapping language and we define how the generic RDF Mapping Language (RML)[4] [10], can properly handle such input sources.

The remainder of the paper is structured as follows. Section 2 elucidates the access and retrieval steps required to obtain the data sources whose semantic representation is desired. Section 3 reviews related works. Section 4 describes machine processable service and dataset descriptions for different cases and Section 5 provides details regarding how RML was extended to take them into consideration. Finally, Section 6 concludes with the outcomes of this work.

2. ACCESS, RETRIEVE AND MAP DATA TO THE RDF DATA MODEL

In this section, we explain the access and retrieval steps required to obtain the data whose semantic representation is desired. Figure 1 illustrates how data is accessed and retrieved from their original repositories and how further data fractions are extracted to finally obtain the desired RDF.

Data is stored in different repositories residing sometimes in different locations. Those repositories can be found e.g., locally, on a network, or on the Web. For instance, data can be available as raw files, databases or Web resources, or files listed in catalogues. To retrieve data from a repository, an *access interface* is required (Step 1) to handle the interaction. Data can be approached using diverse interfaces. For instance, database connectivity, such as Open DataBase Connectivity (ODBC) to access data residing in a database. However, data on the Web can also be retrieved using different interfaces, such as Web APIs or Web services.

Once the *retrieved data* is obtained (Step 2), from one or more repositories, one or more *Logical Sources* are formed. Such a *Logical Source* contains data in a certain structure and format, e.g., CSV, XML or JSON. This data source is what mapping languages, such as RML, consider for the mapping definitions. How this data source is retrieved is out of scope for vocabularies focused on specifying the mapping definitions. If the original repository is a raw file, the *Logical Source* may coincide. Further data fragmentation and extraction requires references relevant to the data format (i.e., its corresponding *Reference Formulation*).

As *mapping definitions* are meant to be applied recursively to data fragments extracted from the *Logical Source*, an *iterator* is required. The iteration pattern is also defined in a formulation relevant to the *Logical Source*. The *iterator* runs over the *Logical Source*, extracting data fragments (Step 3). For instance, an *iterator* running over a CSV file extracts a row of the CSV at each iteration. In case the iteration pattern applies to the complete *Logical Source*, the *Iteration* fragment coincides with the *Logical Source*.

For each *Iteration* further data fragmentation occurs (Step 4) to extract the exact *Data fraction(s)* used to instantiate a *Term Map* which, in its turn, generates the corresponding RDF term. For the aforementioned CSV example, such a data fraction is the value of a column from a given row extracted at a certain *Iteration*. At the end, the corresponding RDF representation of the *Logical Source* is obtained (Step 5).

3. STATE OF THE ART

To the best of our knowledge, there is no mapping solution that takes into consideration diverse dataset and services descriptions to access the data whose RDF representation is desired. Most existing solutions consider data derived from a certain source format and from a given input. In this section, we outline mapping languages (Section 3.1) and vocabularies for dataset access and service descriptions (Section 3.2).

[1] http://www.w3.org/TR/vocab-dcat/
[2] http://www.w3.org/ns/hydra/spec/latest/core/
[3] http://www.w3.org/TR/sparql11-service-description/
[4] http://rml.io/

3.1 Mapping Languages

For relational databases, different mapping languages are defined [11]. Indicatively mentioned, the Triplify [2], which is based on mapping HTTP-URI requests onto relational database queries, and the Sparqlification Mapping Language (SML) [22] which declaratively defines mappings based on SQL views and SPARQL construct queries, do not specify how the input data source is retrieved from the corresponding database within the mapping definitions. Among those language, only D2RQ [6], which is described in more details in Section 4.4, defines how the database connectivity should be specified. To the contrary, the W3C recommended R2RML [7], does not provide any database connectivity descriptions, as it considers such description out of the vocabulary scope.

Mapping languages were also defined to support conversion from data in CSV and spreadsheets to the RDF data model. XLWrap's mapping language [15] maps data in various spreadsheets to RDF, without describing alternative access descriptions. A file is always the expected data source which is specified within the mapping definition as follows `[ ] xl:fileName "files/example.xls"`. Similarly, TARQL [5], that follows a query-based approach, considers CSV files as input. Such a file is also defined within the query, which acts as mapping definition. In TARQL language, the mapping definitions have SPARQL syntax, thus the input CSV file is defined as follows `SELECT ... FROM <file:example.csv>`. Last, the declarative OWL-centric Mapping Master's M^2 [21] language, that maps data from spreadsheets into OWL, does not specify at all within the mapping definitions the input source.

A larger variety of solutions exist to map data in XML format to RDF, but tools mostly rely on existing XML solutions, such as XSLT (e.g., Krextor[5] and AstroGrid-D[6]), XPath (e.g., Tripliser[7]) and XQuery (e.g., XSPARQL[8]). None of them though defines neither how the input source should be specified, nor has RDF syntax which would allow them to be combined with dataset and service access descriptions.

Last, among the tools that provide mappings from different source formats to the RDF data model, e.g., Datalift[9], OpenRefine[10], RDFizers[11] or Virtuoso Sponger[12], none relies on independent generic mapping definitions. Instead those tools employ separate source-centric approaches for each of the formats they support which are hard-coded in the corresponding implementation. The only generic language that exists and allows any type of input source is RML [10], which is described in more details in Section 5.1.

3.2 Dataset and Service Descriptions

Different dataset and service descriptions exist, which describe how to access data. Taking advantage of them, we aim to reuse existing vocabularies, which we summarize and discuss in the following paragraphs.

Dataset descriptions could refer to data catalogues or to Linked Data sets. In the former case, the W3C-recommended vocabulary, DCAT [18], is defined which is more thoroughly described in Section 4.1. In the later case, the VOID vocabulary [1] is considered which defines how to describe metadata for RDF datasets to improve their discoverability. Among the metadata which can be specified with the VOID vocabulary, it is also *access metadata*. The VOID vocabulary allows to specify as access interface (i) SPARQL endpoints, (ii) RDF data dumps, (ii) root resources, (iv) URI lookup endpoints and (v) OpenSearch description documents. Dataset descriptions could also refer to a specific type of data, e.g., tabular data. In this context, the CSV on the Web Working Group[13] aims to define a case-specific metadata vocabulary for Tabular data on the Web [23] which, at its current state, only allows data dumps as access interface.

As far as service descriptions is concerned, and in respect to accessing data in RDF syntax, besides the VOID vocabulary, there is the W3C recommended SPARQL-SD [24], which is described in more details in Section 4.3. Regarding database connectivity, there are no dedicated vocabularies. Descriptions in the frame of mapping languages, e.g., D2RQ, which is also described in more details in Section 4.4, prevailed so far. However, regarding Web APIs and Services, different vocabularies were defined. In the case of Web APIs, there is no W3C-recommended vocabulary. Thus, in Section 4.2, we consider and describe in more details the Hydra vocabulary.

The Web Service Description Language (WSDL) [4] describes the possible interactions, messages and the abstract functionality provided by Web services. [12] describes its representation in RDF and in OWL, as well as a mapping procedure for transforming WSDL descriptions into RDF. Semantic Annotations for WSDL (SAWSDL) [14] was one of the first attempts to offer semantic annotations for Web services. Later on, an adaptation for generic HTTP interfaces was proposed [19]. The OWL for Services (OWL-S) [20], the Web Service Modeling Ontology (WSMO) [8] and the WSMO-Lite [25] are alternative ontologies, defined for modelling Web services. The OWL-S ontology also focuses on input and output parameters, as SAWSDL. The WSMO ontology is an alternative to OWL-S, although there are substantial differences between the two approaches [17]. The WSMO ontology employs a single family of layered logic languages [9]. However, when expressed in RDF syntax, WSMO expressions become similarly unintegrated and hence not self-descriptive as OWL-S expressions. The WSMO-Lite ontology extends SAWSDL with conditions and effects. hRESTS [13] uses microformats to add machine-processable information to human-readable documentation, while its ontology[14] extends the WSMO-Lite ontology[15]. Last, MicroWSMO extends hRESTS and adopts the WSMO-Lite service ontology for expressing concrete semantics. For our purposes, we mostly need the interface description part of the above possibilities, since our goal is access to the services rather than, for instance, composition.

4. DESCRIBING INTERFACES TO ACCESS HETEROGENEOUS DATA SOURCES

Even though the barrier of uniformly mapping heterogeneous data to the RDF data model has been overcome, with uniform, machine-interpretable mapping definitions and case specific references to the input data source, depending on its

[5] https://trac.kwarc.info/krextor/

[6] http://www.gac-grid.de/project-products/Software/XML2RDF.html

[7] http://daverog.github.io/tripliser/

[8] http://www.w3.org/Submission/xsparql-language-specification/

[9] http://datalift.org/

[10] http://openrefine.org/

[11] http://simile-widgets.org/wiki/Main_Page

[12] http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtSponger

[13] http://www.w3.org/2013/csvw/wiki/Main_Page

[14] http://www.wsmo.org/ns/hrests/

[15] http://www.wsmo.org/ns/wsmo-lite/

format, data access and retrieval remains hard-coded in the implementation. Accessing data might be straightforward, as in the case of files locally stored. However, in most cases, dedicated interfaces are required. In any case, corresponding vocabularies can describe how to access the underlying data. Such vocabularies may refer to: (i) the dataset's metadata, (ii) Hypermedia-driven Web APIs or services, (iii) SPARQL services, and (iv) Database connectivity frameworks.

In the previous section (Section 3), we review vocabularies describing interfaces for accessing datasets and services which enable agents to retrieve the underlying data. For each type of interface, a corresponding W3C-recommended vocabulary is described below. In case there is no such vocabulary, a widely-used one is taken into account. The list is not exhaustive, it rather has an indicative exemplary purpose, aiming to capture the most common cases. Any of the dataset or service descriptions could be replaced by other corresponding ones and new can be considered.

4.1 Metadata for the Access Interface

Data can be published either independently, as data dumps, or in the frame of a data catalogue. It provides machine-readable metadata that enables applications to easily consume them. It does not make any assumptions about the format of the datasets described in a catalog; format-specific information is out of scope. The DCAT namespace is http://www.w3.org/ns/dcat# and the preferred prefix is *dcat*.

The DCAT vocabulary defines *dcat:Catalog* that represents a dataset catalog, *dcat:Dataset* that represents a dataset in the catalog, while *dcat:Distribution* represents an accessible form of a dataset, e.g., a downloadable file, an RSS feed or a Web service that provides the data. DCAT considers as a dataset a collection of data, published or curated by a single agent, and available for access or download in one or more formats. Thus, a certain distribution is the minimum that a mapping processor requires. Directly downloadable distributions contain a *dcat:downloadURL* reference, for instance:

```
@prefix dcat: <http://www.w3.org/ns/dcat#>.
<#DCAT_source>
  a dcat:Dataset;
  dcat:distribution [
    a dcat:Distribution;
    dcat:downloadURL <http://example.org/file.xml>].
```

Listing 1: DCAT access metadata description

A *dcat:Distribution* might not be directly downloadable though. For instance, a dataset might be available via an API and the API, in its turn, can be defined as an instance of a *dcat:Distribution*. In this case, it is recommended to use *dcat:accessURL* instead of *dcat:downloadURL*. However, access-specific properties, e.g., for API descriptions, is not defined by DCAT itself. Thus a client does not know how to interact with the mentioned interface, the API in this case. Due to DCAT shortcoming to entirely describe indirectly accessed Web sources, other vocabularies focused on describing specific interfaces could be considered instead.

4.2 Hypermedia-Driven Web APIs

For the description of hypermedia-driven Web APIs, the Hydra Core Vocabulary [16], a lightweight vocabulary used to specify concepts commonly used in Web APIs, is published by the Hydra W3C Community Group[16]. The Hydra vocabulary provides machine-processable descriptions which enable a server to advertise valid state transitions to a client. The server is decoupled from the client which can use this information to construct valid HTTP requests to retrieve the data. The Hydra namespace is http://www.w3.org/ns/hydra/core# and the preferred prefix *hydra*.

An instance of the *hydra:ApiDocumentation* class describes a Web API, by providing its title, short description, main entry point and additional information about status codes that might be returned. The Hydra vocabulary enables the API's main entry point to be discovered automatically, when it is not known or specified, if the API publisher marks his responses with a special HTTP link header. A client looks for a link header with a relation type *hydra:apiDocumentation* and, this way, obtains a *hydra:ApiDocumentation* defining the API's main entry point.

The *dcat:accessUrl* of a *dcat:Distribution* instance can point to a resource described with the Hydra vocabulary, informing potential agents how valid HTTP requests should be performed. The Hydra vocabulary can be used both to describe (i) static IRIs, and (ii) dynamically generated IRIs, e.g., Listing 2. A template valued IRI containing variables, is described as a *hydra:IriTemplateMapping* instance whose values depend on information only known by the client.

```
@prefix hydra: <http://www.w3.org/ns/hydra/core#>.
<#API_source>
  a hydra:IriTemplate
  hydra:template "https://api.twitter.com/1.1/followers/ids.
      json?screen_name={name}";
  hydra:mapping [
    a hydra:IriTemplateMapping;
    hydra:variable "name";
    hydra:required "true" ].
```

Listing 2: Template-valued Web API description

Web APIs often split a collection of data into multiple pages. In Hydra, this is described with an instance of the *hydra:PagedCollection* that contains information regarding the total number of items, the number of items per page and the first, the next and the last page. An example instance could be as follows:

```
@prefix hydra: <http://www.w3.org/ns/hydra/core#> .
<#APIpaged_source>
  a hydra:PagedCollection;
  hydra:apiDocumentation <#HydraDocumentation>;
  hydra:itemsPerPage "100";
  hydra:firstPage "/comments?page=1";
  hydra:lastPage "/comments?page=10" .
```

Listing 3: Hydra Paged Collection description

Web services played an important part in the initial Semantic Web vision [3]. However, they were surpassed in popularity by Web APIs and, at the moment, most of the Web-based solutions prefer the later. Thus, a detailed example for Web Service descriptions is not provided, even though such a description could equally be considered.

4.3 SPARQL Services

For the description of SPARQL endpoints, W3C recommends the SPARQL Service Description vocabulary (SPARQL-SD) [24]. SPARQL-SD provides a list of features of a SPARQL service and their descriptions, made available via the SPARQL 1.1 Protocol for RDF. The SPARQL-SD namespace is http://www.w3.org/ns/sparql-service-description# and the preferred prefix is *sd*.

[16] http://www.w3.org/community/hydra/

An instance of *sd:Service* represents a SPARQL service made available via the SPARQL protocol. A *sd:Service* refers to a default dataset, described as an instance of the *sd:Dataset* that represents an RDF dataset comprised of a default graph (an instance of *sd:Graph*) and zero or more named graphs (an instance of *sd:NamedGraph*). A collection of graphs is described as instances of *sd:GraphCollection*. Last, SPARQL-SD defines *sd:Language* whose instances represent one of the SPARQL languages (e.g., *sd:SPARQL11Query*).

```
@prefix sd:
  <http://www.w3.org/ns/sparql-service-description#>.
<#SPARQL_source>
  a sd:Service;
  sd:endpoint <http://dbpedia.org/sparql/>;
  sd:supportedLanguage sd:SPARQL11Query;
  sd:resultFormat
    <http://www.w3.org/ns/formats/SPARQL_Results_XML>.
```

Listing 4: SPARQL Service Description

Similarly to *hydra:IriTemplate*, a *sd:Service* instance could be used to clarify *dcat:accessUrl*, allowing potential agents to know how to perform the corresponding HTTP requests. In the same way, *void:endpoint* could be considered in the case of datasets published with metadata described using the W3C-recommended VOID vocabulary.

4.4 Database Connectivity

For the description of database connectivity, corresponding descriptions from the D2RQ mapping language [6] can be considered for accessing databases with the JDBC framework. D2RQ is a declarative mapping language for describing the relation between a database schema and RDFS vocabularies or OWL ontologies. The D2RQ namespace is http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1# and the preferred prefix *d2rq*.

A *d2rq:Database* instance defines a JDBC connection to a local or remote relational database. Instances of d2rq:Database, annotated with its properties to specify the JDBC connection properties, can be used to describe the access to a database. An instance of such database description is as follows:

```
@PREFIX d2rq:
  <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#>.
<#DB_source>
  a d2rq:Database;
  d2rq:jdbcDSN "jdbc:mysql://localhost/example";
  d2rq:jdbcDriver "com.mysql.jdbc.Driver";
  d2rq:username "user";
  d2rq:password "password".
```

Listing 5: D2RQ database connectivity description

The D2RQ database connectivity description is focused on databases with JDBC-based connectivity and serves the needs of an exemplary case of this work. Other vocabularies describing the JDBC framework or vocabularies for other interfaces of database connectivity can be considered as well.

5. ALIGNING DATA ACCESSING AND MAPPING TO RDF DESCRIPTIONS

In this section, we define in details how heterogeneous dataset and service descriptions can be taken into consideration to access data and instantiate RML Logical Sources. Those Logical Sources contain data whose representation in RDF syntax is desired. For our use case, we align the aforementioned descriptions with the RML mapping language [10]. In Section 5.1, we introduce the language, and in Section 5.2, we concretely define how RML Logical Sources are obtained via such dataset and service descriptions. Finally, in Section 5.3 we introduce required extension to the RML mapping language to entirely support such logical sources. Detailed documentation regarding access interfaces supported by RML is available at http://rml.io/RMLdataRetrieval.

5.1 RML

RML [10] extends R2RML [7], the W3C-recommended mapping language for defining mappings of data in relational databases to the RDF data model, by broadening its scope. RML covers also mappings from data sources in different (semi-)structured formats, such as CSV, XML, and JSON.

```
@prefix rr:     <http://www.w3.org/ns/r2rml#>.
@prefix rml:    <http://semweb.mmlab.be/ns/rml#>.
@prefix foaf:   <http://xmlns.com/foaf/0.1/>.

<#Person> rml:logicalSource <#InputX> ;
  rr:subjectMap [ rr:template "http://ex.com/{ID}"];
  rr:predicateObjectMap[
   rr:predicate foaf:account;
   rr:objectMap [ rr:parentTriplesMap <#TwitterAcount> ] ].

<#TwitterAcount> rml:logicalSource <#InputY>;
  rr:subjectMap [ rr:template "http://ex.com/{account_ID}"];
  rr:predicateObjectMap [
   [ rr:predicate foaf:accountName;
     rr:objectMap [ rml:reference "name"] ],
   [ rr:predicate foaf:accountServiceHomepage;
     rr:objectMap [ rml:reference "resource"] ].
```

Listing 6: RML mapping definitions

RML documents contain rules defining how the input data is represented in RDF. An RML document (see Listing 6) contains one or more Triples Maps (line 5 and 11). A Triples Map defines how triples are generated and consists of three main parts: the Logical Source, the Subject Map and zero or more Predicate-Object Maps. The Subject Map (line 6 and 12) defines how unique identifiers (URIs) are generated for the resources and is used as the subject of all RDF triples generated from this Triples Map. A Predicate-Object Map (line 7 and 13) consists of Predicate Maps, which define the rule that generates the triple's predicate (line 8, 14 and 16) and Object Maps (line 14 and 16) or Referencing Object Maps (line 8), which define how the triple's object is generated.

```
<#InputX>
  rml:source "/.../.../file.csv";
  rml:referenceFormulation ql:CSV.
```

Listing 7: RML Logical Source definition – local file

A Logical Source (see Listing 7) is used to determine the input source (line 2) with the data to be mapped and how to refer to them (line 3). RML deals with different data serialisations which use different ways to refer to their elements/objects. RML considers that any reference to the Logical Source should be defined in a form relevant to the input data, e.g., XPath for XML files or JSONpath for JSON files. To this end, the Reference Formulation (line 3) declaration is stated indicating the formulation (for instance, a standard or a query language) used to refer to its data. At the current version of RML, the `ql:CSV`, `ql:XPath`, `ql:JSONPath` and `ql:CSS3` Reference Formulations are predefined, but not limited.

5.2 Dataset and Service Access Descriptions as RML Logical Sources

RML provides a generic way to define the mappings that is easily transferable to cover references to other data structures. RML needs to deal with different data serialisations which use different ways to refer to their data fragments. Since RML aim is generic, there is no uniform way of referring to these data fragments. RML considers that any reference to the source should be defined in a form relevant to the input data, e.g., XPath for XML files or JSONpath for JSON files. This is defined using the Reference Formulation (rml:referenceFormulation) declaration that indicates the formulation (for instance, a standard or a query language) used to refer to source's data fragments.

However, the RML specification is focused on the rules defining how to generate the RDF data. RML considers a given original data input, but the way this input is retrieved remains out of scope, in the same way as it remains out of scope for R2RML specification how the SQL connection is established. The input data is specified with the Logical Source, as well as how to refer to this data, but not how to access and retrieve this data. Namely, the Logical Source consists of some data without further defining how to retrieve the data.

The access descriptions, that advertise services or datasets, could be considered as the Triples Map's Source (*rml:source*). For instance, the Logical Source specified at Listing 6 for the `<#Person>` Triples Map, instead of having been specified as a local file (see Listing 7), it could have been published on a data catalogue and, thus, it is an instance of *dcat:Distribution*. The corresponding description then would be as follows:

```
@prefix rml:    <http://semweb.mmlab.be/ns/rml#>.
@prefix dcat:   <http://www.w3.org/ns/dcat#> .

<#InputX>
  rml:source [ a dcat:Distribution;
    dcat:downloadURL "http://ex.com/file.csv" ];
  rml:referenceFormulation ql:CSV .
```

Listing 8: Data dump in catalogue as Input Source

For the other Triples Map, `<#TwitterAccount>`, the data to be mapped might be derived from a user's twitter account, and could have been stored locally in a file retrieved at some point from the Twitter API, or the request could have been hard-coded in the implementation. Nevertheless, the required request could have just been described using the Hydra vocabulary or could have been provided using directly the resource advertising the API. In the aforementioned example of Listing 6, the Logical Source for the `<#TwitterAccount>` Triples Map could have been described as follows:

```
@prefix hydra: <http://www.w3.org/ns/hydra/core#>.

<#InputY> a hydra:IriTemplate
  hydra:template "https://api.twitter.com/1.1/followers/ids.
      json?screen_name={name}";
  hydra:mapping [
    hydra:variable "name";
    hydra:required "true" ].
```

Listing 9: Web API as Input Source

5.3 RML Referencing Object Map and Heterogeneous Data Retrieval

The use of data derived from such a *Logical Source*, formed by instantiating an access description, is straightforward in most cases. Dataset and service descriptions either are derived from *data owners/publishers* or explicitly defined by *data publishers/consumers*. Mapping processors take them into consideration to be informed regarding how to access the data and instantiate the *Logical Sources*. The access description might be static or dynamic. If dynamically created, it is often required to instantiate a template, e.g., a URI template or a SQL/SPARQL query template. The values to instantiate the template might be provided by the user who executes the mapping or the variables might be replaced with values derived from another input source, as it occurs in the case of *Referencing Object Maps*.

Binding Condition

A Referencing Object Map (see Listing 10, `<#TwitterAccount>` was specified at Listing 6) allows using the subject of another Triples Map(line 13) as the objects generated by a Predicate Object Map and the two Triples Maps may be based on different Logical Sources. A Referencing Object Map (line 8) might have a *template-valued* input source (line 15) that requires one or more values to be filled with.

We introduced the Binding Condition to address this issue. The Binding Condition specifies how the Logical Source of the Referencing Object Map is instantiated either with value(s) retrieved from the input source that is currently mapped, specified with an rml:reference or a rr:template, or with a constant value, specified with a rr:constant. In the first case, a reference (line 10) that exists in the Logical Source of the Triples Map that contains the Referencing Object Map is provided. In the later case, a constant value can be provided. The value is bind a variable that is specified with crml:variable. If the Referencing Object Map's Logical Source has more than one variables required, equal number of Binding Conditions is expected. Detailed documentation regarding the *RML Binding Condition* is available at http://rml.io/RMLconditions.

```
@prefix rr:     <http://www.w3.org/ns/r2rml#>.
@prefix rml:    <http://semweb.mmlab.be/ns/rml#>.
@prefix crml:   <http://semweb.mmlab.be/ns/crml#>.

<#Person>
  rr:predicateObjectMap [ rr:predicate foaf:account;
    rr:objectMap [
      rr:parentTriplesMap <#TwitterAcount> ;
      crml:binding [
        rml:reference "id" ;
        crml:variable "name" ]]].

<#InputY> rml:source [
  a hydra:IriTemplate
  hydra:template "https://api.twitter.com/1.1/followers/ids.
      json?screen_name={name}";
  hydra:mapping [
    hydra:variable "name";
    hydra:required "true" ] ];
  rml:referenceFormulation ql:JSONPath.
```

Listing 10: RML Binding Condition

5.4 Implementation

An RMLProcessor can be implemented using two alternative models [10]: (i) *mapping-driven*, where the processing is driven by the mapping module; or (ii) *data-driven*, where the processing is driven by the extraction module. When the RML mappings are processed, the mapping module deals with the mapping definitions execution, while the extraction module deals with the target language expressions (expression using the corresponding *Reference Formulation*).

On the *mapping-driven* occasion, the *mapping module* requests an extract of data from the *extraction module*, considering the iterator specified at the *Logical Source*. On the *data-driven* occasion, an extract of data is passed to the *mapping module*, which performs the applicable mappings.

A new additional independent *access and retrieval module* is introduced to deal with the retrieval of data that form the *Logical Source*. The *retrieval module* relies on the access description to retrieve the *Logical Source*. The access description can be provided either by the *data owner/publisher* or by the *data consumer/publisher*. Moreover, if the access description is dynamically generated either user input is taken into consideration to bind the template variables with values, or values derived from another *Logical Source* Overall, none of the two aforementioned cases (*mapping-* or *data-driven*) is affected by the way the data is accessed and retrieved. A separate project, *RMLDataRetrieval*, is introduced as part of the RMLProcessor[17]. The *RMLDataRetrieval* project is included in the RMLProcessor and currently supports most of the access interfaces described in Section 4.

5.5 Discussion

Being able to consider diverse access interfaces facilitates the description of the interaction models while the original data remains independent, interoperable and granular. In the same time, the alignment of dataset and service descriptions with the mapping definitions as proposed in this work, demands certain clarifications. Firstly, it is required to address the cases where both the *data publishers/consumers* provides access descriptions and the *data publishers/owners*. Then, it is required to elucidate how the mapping definitions should be defined in the case of relational databases where the database connectivity is desired to be specified in comparison with the R2RML compliant mapping definitions that do not specify the database connectivity. Last, the role of accessing data already in RDF, e.g., with SPARQL-SD or VOID descriptions, as input *Logical Sources* should be clarified.

Published vs. Defined Data Access Description

If the service provides access metadata, the *data publisher/-consumer* can just point to the resource that describes them. If not, the minimum required information for each access interface should be defined. In case the data access is described by the RDF *data publisher/consumer*, its description prevails over the one provided by the *data publisher/owner*. For instance, in the case of a *hydra:PagedCollection* instance, the *data publisher/consumer* might define at the data access description that a hundred items per page should be returned and five pages should be taken into consideration. If the publishing service returns an answer that consists of ten pages of data, only the five of them should be mapped. If the *data publisher/consumer* does not specify the pages, all of them will be considered for mapping.

Database Connectivity Description with RML Logical Source and R2RML Logical Table

A Logical Source (see Listing 7) extends R2RML's Logical Table and is used to determine the input source with the data to be mapped and how to refer to them. The R2RML Logical Table definition determines a database's table, using the Table Name. Nevertheless, how the connection to the database is achieved is not specified at the R2RML specification, since it remains out of its scope. Moreover, R2RML is specific for relational databases (SQL), while a D2RQ description may refer to other databases using the JDBC framework too, or in general, any other database with its corresponding description of the framework may be considered. In the case of an SQL query against the table *DEPT* of a database for instance, the R2RML *Logical Table* would have been defined as follows:

```
[ ] rr:logicalTable [ rr:sqlQuery """
  SELECT DEPTNO, DNAME, LOC,
  (SELECT COUNT(*) FROM EMP WHERE EMP.DEPTNO=DEPT.DEPTNO)
  AS STAFF FROM DEPT; """ ].
```

Listing 11: R2RML Logical Table

However, if a database is specified, the *Logical Table* should be superseded by its broader *Logical Source* and the corresponding database connectivity description should be provided, as follows (`<#DB_source>` was defined at Listing 5):

```
[ ] rml:logicalSource [
  rml:query """
    SELECT DEPTNO, DNAME, LOC,
    (SELECT COUNT(*) FROM EMP WHERE EMP.DEPTNO=DEPT.DEPTNO)
    AS STAFF FROM DEPT; """ ;
  rml:source <#DB_source> ].
```

Listing 12: RML Logical Source for Database Input

SPARQL Service as Logical Source

Having a SPARQL-SD as *Logical Source* might seem contradictory, as the data it contains are already semantically annotated and, thus, it is not required to be mapped to the RDF data model. However, in the cases a resource is already defined and assigned a URI and no new URI is willing to be generated, it is rather preferred to point to this resource.

```
@prefix rr:     <http://www.w3.org/ns/r2rml#>.
@prefix rml:    <http://semweb.mmlab.be/ns/rml#>.
@prefix sd:
  <http://www.w3.org/ns/sparql-service-description#>.

<#Address> rr:predicateObjectMap [
  rr:predicate ex:country ;
  rr:objectMap [ rr:parentTriplesMap <#Country> ] ].

<#Country> rml:logicalSource [
  rml:query """ SELECT distinct ?Concept
    WHERE { ?Concept a dbpedia-owl:Country;
                     rdfs:label "Belgium"@en} """;
  rml:source <#DBPedia> ];
  rr:subjectMap [ rr:termType rr:IRI ;
    rml:reference "/sparql/results/result/binding/uri" ].

<#DBPedia>
  sd:endpoint <http://dbpedia.org/sparql/>;
  sd:supportedLanguage sd:SPARQL11Query;
  sd:resultFormat
    <http://www.w3.org/ns/formats/SPARQL_Results_XML>.
```

Listing 13: SPARQL Endpoint for Input Source

For instance, there is a CSV file containing some data related to addresses, and among others, there is a column with country names and a certain cell might contain e.g., Belgium. Instead of generating a new resource with a new unique URI, the DBpedia URI could be considered. This can be achieved with a *Referencing Object Map* whose *Logical Source* is the result of executing a query against, for instance, DBpedia endpoint, whose access description, in its turn, is defined as a *sd:Service* instance (see Listing 13).

[17] https://github.com/RMLio/RML-Processor

6. CONCLUSIONS AND FUTURE WORK

In this paper, we introduce an approach that exploits vocabularies originally used to advertise services or datasets, to define how to access data whose semantic representation is desired. While, machine-interpretable descriptions of data access remain independent, their alignment with uniform machine-interpretable mapping definitions leads to a granular but robust solution which further automates and facilitates the generation of RDF representations.

7. ACKNOWLEDGEMENTS

This paper's research activities were funded by Ghent University, iMinds, the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research-Flanders (FWO-Flanders), and the EU.

8. REFERENCES

[1] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing Linked Datasets with the VoID Vocabulary. W3C Interest Group Note, Mar. 2011. http://www.w3.org/TR/void/.

[2] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: Light-weight Linked Data Publication from Relational Databases. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09. ACM, 2009.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 2001.

[4] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, Mar. 2001. http://www.w3.org/TR/wsdl.

[5] R. Cyganiak. Tarql – SPARQL for Tables: Turn CSV into RDF using SPARQL syntax. Technical report, Jan. 2015. http://tarql.github.io/.

[6] R. Cyganiak, C. Bizer, J. Garbers, O. Maresch, and C. Becker. The D2RQ Mapping Language. Technical report, Mar. 2012. http://d2rq.org/d2rq-language.

[7] S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF Mapping Language. Working Group Recommendation, W3C, Sept. 2012. http://www.w3.org/TR/r2rml/.

[8] J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, U. Keller, M. Kifer, B. KÃűnig-Ries, J. Kopecky, R. Lara, H. Lausen, E. Oren, A. Polleres, D. Roman, J. Scicluna, and M. Stollberg. Web Service Modeling Ontology (WSMO). W3C Member Submission, June 2005. http://www.w3.org/Submission/WSMO/.

[9] J. de Bruijn, D. Fensel, U. Keller, M. Kifer, H. Lausen, R. Krummenacher, A. Polleres, and L. Predoiu. Web Service Modeling Language (WSML). W3C Member Submission, June 2005. http://www.w3.org/Submission/WSML/.

[10] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Workshop on Linked Data on the Web*, 2014.

[11] M. Hert, G. Reif, and H. C. Gall. A comparison of RDB-to-RDF mapping languages. I-Semantics '11. ACM, 2011.

[12] J. Kopecký. Web Services Description Language (WSDL) Version 2.0: RDF Mapping. W3C Working Group Note, June 2007. http://www.w3.org/TR/wsdl20-rdf/.

[13] J. Kopecký, K. Gomadam, and T. Vitvar. hrests: An HTML Microformat for Describing RESTful Web Services. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*. IEEE Computer Society, 2008.

[14] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11, 2007.

[15] A. Langegger and W. Wöß. XLWrap – Querying and Integrating Arbitrary Spreadsheets with SPARQL. In *Proceedings of 8th ISWC*. Springer, 2009.

[16] M. Lanthaler. Hydra Core Vocabulary. Unofficial Draft, June 2014. http://www.hydra-cg.com/spec/latest/core/.

[17] R. Lara, D. Roman, A. Polleres, and D. Fensel. A Conceptual Comparison of WSMO and OWL-S. In *Web Services*, volume 3250 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004.

[18] F. Maali and J. Erickson. Data Catalog Vocabulary (DCAT). W3C Recommendation, Jan. 2014. http://www.w3.org/TR/vocab-dcat/.

[19] M. Maleshkova, J. Kopecký, and C. Pedrinaci. Adapting SAWSDL for Semantic Annotations of RESTful Services. In *On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, volume 5872 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009.

[20] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. W3C Member Submission, Nov. 2004. http://www.w3.org/Submission/OWL-S/.

[21] M. J. O'Connor, C. Halaschek-Wiener, and M. A. Musen. Mapping Master: a flexible approach for mapping spreadsheets to OWL. Proceedings of 9th ISWC, 2010.

[22] C. Stadler, J. Unbehauen, P. Westphal, M. Ahmed Sherif, and J. Lehmann. Simplified RDB2RDF Mapping. In *Workshop on Linked Data on the Web*, 2015.

[23] J. Tennison, G. Kellogg, and I. Herman. Model for Tabular Data and Metadata on the Web. W3C Working Draft, Apr. 2015. http://www.w3.org/TR/2015/WD-tabular-data-model-20150416/.

[24] G. Todd Williams. SPARQL 1.1 Service Description. W3C Recommendation, Mar. 2013. http://www.w3.org/TR/sparql11-service-description/.

[25] T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel. WSMO-Lite Annotations for Web Services. In *The Semantic Web: Research and Applications*, volume 5021 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008.