Ademar Crotti Junior, Christophe Debruyne, Rob Brennan, and Declan O'Sullivan. An evaluation of uplift mapping languages. International Journal of Web Information Systems, 13(4):405–424, 2017

An Evaluation of Uplift Mapping Languages

ABSTRACT

Purpose: Typically tools that map non-RDF data into RDF format rely on the technology native to the source of the data when data transformation is required. Depending on the data format, data manipulation can be performed using underlying technology, such as RDBMS for relational databases or XPath for XML. For CSV/Tabular data there is no such underlying technology, and instead it requires either a transformation of source data into another format or pre/post-processing techniques. In this paper we evaluate the state of the art in CSV uplift tools. Based on this evaluation, a method that incorporates data transformations into uplift mapping languages by means of functions is proposed and evaluated.

Design/methodology/approach: In order to evaluate the state of the art in CSV uplift tools we present a comparison framework and have applied it to such tools. A key feature evaluated in the comparison framework is data transformation functions. We argue that existing approaches for transformation functions are complex – in that a number of steps and tools are required. Our proposed method, FunUL, in contrast, defines functions independent of the source data being mapped into RDF, as resources within the mapping itself.

Findings: The approach was evaluated using two typical real world use cases. We have compared how well our approach and others (that include transformation functions as part of the uplift mapping) could implement an uplift mapping from CSV/Tabular into RDF. This comparison indicates that our approach performs well for these use cases.

Originality/value: This paper presents a comparison framework and applies it to the state of the art in CSV uplift tools. Furthermore, we describe FunUL, which unlike other related work, defines functions as resources within the uplift mapping itself, integrating data transformation functions and mapping definitions. This makes the generation of RDF from source data transparent and traceable. Moreover, since functions are defined as resources, these can be reused multiple times within mappings.

Keywords: Linked Data, Mapping, Data Transformation.

1. INTRODUCTION

Significant amounts of data on the Web still resides in formats other than the Resource Description Framework (RDF) (Klyne et al., 2004) data model, currently being advocated by the W3C community as the means to enable data exchange on the Web, and a variety of innovative applications, such as data integration and others (Hitzler et al., 2009). CSV/Tabular data (even though the delimiter is different, for conciseness we refer to such data as CSV data for the rest of this paper) is commonly used for data exchange on the Web, but the semantics of the data are not made explicit in the data format. In contrast, RDF provides one means to publish data and its meaning.

The process of converting data into RDF is called *uplift*. As many solutions have been proposed to uplift CSV data into RDF, we have developed a framework to compare these. For this comparison framework, we have drawn inspiration from a similar framework to evaluate the mapping of relational databases into RDF presented in Hert et al. (2011). We have applied our comparison framework to those state-of-start uplift tools that have support for CSV data.

One of the key features evaluated in our framework is the support for transformation functions, as data manipulation is typically needed during the uplift process (Purohit et al., 2016). These functions can be used to capture both domain knowledge (e.g., transforming units) and other, more syntactic, data manipulation tasks (e.g., transforming values to create valid IRIs). For some data formats this can be more-or-less straightforward. For example, with uplift tools for relational databases, such as R2RML (Das et al., 2012) implementations, one can rely on SQL to provide the necessary transformation functions, whereas with RML (Dimou et al., 2014), an R2RML extension with support for various data formats, XPath is used to transform XML data and JSONPath is used for JSON data. In many cases, however, relying on underlying technology to undertake data transformations might not be possible (Debruyne and O'Sullivan, 2016). One such case is for CSV data, where there is no such underlying technology. The general approach to manipulate CSV data is the transformation of source data into another format or the use of pre/post-processing techniques. The use of these techniques, however, increases complexity – in relation to the number of steps and tools involved – and renders the data process pipeline less transparent and traceable.

One example of data transformation is the conversion of years. A historical dataset might use BCE/CE notation to refer to years, but an RDF representation of this data may use the datatype `xsd:gYear`, from the XML Schema specification (Biron and Malhotra, 2004), for representing years in a Gregorian calendar. The year "31 BCE" in the dataset would thus need to be transformed into `"-30"^^xsd:gYear`.

To overcome these problems, we have, in previous work (Junior, A. C. et al., 2016a; Junior, A. C. et al., 2016b), proposed a method to include data transformations by the way of functions into Uplift Mapping Languages in a generic, reusable and amenable way, defined as FunUL. We adapt the definition of Uplift Mapping Language from (Bizer and Seaborne, 2004) and define it as a declarative language for mapping non-RDF data sources into RDF vocabularies and OWL ontologies.

FunUL allows data transformation and uplift of data into RDF to happen in a single unified step. In addition, function definitions are reusable, making it possible to use the same function multiple times with different parameters; and traceable, since mapping and function definitions are specified in the same file. It is also possible to annotate the functions with provenance information, descriptions of the transformation and other useful metadata.

FunUL draws inspiration and generalizes ideas from our previous work, R2RML-F (Debruyne and O'Sullivan, 2016), which incorporates functions into R2RML by extending the db2triples [1] R2RML processor. FunUL has been implemented in RML, which also extends db2triples. This RML extension can apply functions to all input data formats supported by RML, including CSV, since our functions are data source technology independent,

Our previous FunUL evaluation (Junior, A. C. et al., 2016b) used our RML extension in a real world dataset. However, RML is not a W3C Recommendation, whereas R2RML is. Moreover, db2triples – and its extensions - are not fully compliant with R2RML's specification, as it does not support named graphs for example. For these reasons, we have developed a new implementation that is fully compliant with the R2RML specification. This implementation has support for CSV data and is used for the evaluations described in this paper.

This paper thus extends upon results presented in the aforementioned previous works. The evaluation (using a new implementation) is applied to two real world datasets, and is presented through a comparison of our approach with the only other approach that we have identified to include data transformation functions as part of the mapping itself, which is KR2RML (Slepicka et al., 2015).

The remainder of this paper is organized as follows: in Section 2 presents the state-of-the-art in CSV uplift. Section 3 shows a comparison and discussion of the tools presented in Section 2. Section 4 presents a method to incorporate functions into mapping languages called FunUL. The evaluation is presented in Section 5. Section 6 concludes the paper.

2. STATE-OF-THE-ART IN CSV UPLIFT

Many solutions have been proposed to uplift CSV data into RDF. This section briefly introduces some of the tools in the state of the art that have support for CSV data.

Uplift techniques can be described at a high level as to the type of approach that they support: mapping languages and additional software support. **Mapping Languages** are declarative languages used to express customized mappings defining how non-RDF data should be represented in RDF. An uplift engine is usually associated with a mapping language, being a software processor that uses the mapping file and the input data to generate an RDF dataset**. Additional Software Support** represents applications that have an interface or API where it is possible to convert data into RDF. Some uplift tools have support for both approaches. Table 1 shows a brief description of each tool and the support provided by them.

Table 1

A framework that allows for a more detailed comparison of these tools is presented in the next section.

3. COMPARING THE STATE-OF-THE-ART IN CSV UPLIFT

In this section, we present a comparison framework to evaluate the state-of-the-art in CSV uplift systems, which was presented in Section 2. Hert et al. (2011) proposed a comparison framework to evaluate relational databases to RDF tools on a feature-by-feature basis. We have adapted this framework into the CSV uplift context by proposing 2 new features. One specific feature for CSV data: filtering; and a second feature for the whole uplift process: reusability. The features of the comparison framework are enumerated as follows; with new features annotated with (N):

- **F1: M:N Relationships.** A CSV dataset may contain column-related information. In this case, two columns are mapped as resources. An uplift tool should support the definition of relationships between resources.
- **F2: Additional Data.** In some cases, it is necessary to provide additional information about resources or the RDF data that will be generated (e.g. provenance information). This feature allows the definition of new additional data to be generated during the uplift process.
- **F3: Select.** A dataset may contain attributes that should not be a part of the RDF representation. This feature allows the selection of attributes or columns from a CSV dataset to be converted into RDF.
- **F4: Filter (N).** A dataset may contain invalid information or specific information that should not be part of the RDF representation. A mapping language should support the definition of filters to decide whether particular information is valid to the RDF representation. For relational databases, this feature is available through SQL queries with a WHERE clause (specifying conditions). R2RML, for instance, prescribes that Term Maps applied on a NULL value do not generate an RDF term. In other words, for R2RML, a NULL value is an indicator that no information should be generated.
- **F5: Literal to IRI.** This feature allows the transformation of literals from a dataset into valid IRI's for the RDF representation. For example, a dataset may contain a literal for an ISSN number that needs to be transformed into a valid IRI.
- **F6: Vocabulary Reuse.** The vocabulary used in an RDF dataset can be created, generated automatically based on the source data, or existing vocabularies can be reused when defining the RDF representation. This feature allows the reuse of such existing vocabularies.
- **F7: Transformation Functions.** Some attributes may require a different representation in RDF (e.g. different unit measurements). Data transformation functions allow data to be manipulated and transformed before generating RDF triples.
- **F8: Datatypes.** CSV data does not contain datatypes. Every value in a CSV file is of type string. This feature allows the attribution of XML datatypes to attributes when mapping data into RDF. This feature does not evaluate the use of specific datatypes.
- **F9: Named Graphs.** Named graphs are RDF datasets identified using an IRI (Hitzler et al., 2009). Applications should support the generation of RDF into specific named graph.

- **F10: Blank Nodes.** Blank nodes are RDF statements with no RDF IRI reference (Hitzler et al., 2009). This feature allows the generation of blank nodes.
- **F11: Reusability (N).** This feature allows the serialization of the uplift process for further reuse.

In order to evaluate the tools presented in **Table** , we have analyzed papers, documentation and tested a working implementation against the features defined in the comparison framework. In other words, we have used the information and implementation available to define a simple uplift process covering the feature. In the analysis of F1, for example, we would examine the paper, documentation and execute the tool trying to define a mapping relating two different resources. We note that we were unable to evaluate xR2RML. The mapping language has support for multiple data formats, but the available implementation only supports relational and NoSQL databases. Table 2 presents our comparison framework applied to CSV uplift tools.

Table 2

3.1 Discussion

This section discusses the state-of-the-art in CSV uplift using our comparison framework. The discussion is structured according to each feature. If an uplift tool is not mentioned then the tool supports the feature.

- **F1 M:N Relationships.** DataOps does not have support for the definition of relationships between related resources. OpenRefine supports this feature by creating a new IRI with an existing IRI value, as it is not possible to select existing IRI resources. DataLift has support for this feature but it can be very complex, due to its modular design, to redefine the RDF dataset after the direct mapping process. Top Braid Composer supports this feature through SPARQL CONSTRUCT queries. Overall, this feature is supported by most tools but some offer more amenable ways to define M:N relationships, as with RML, where identified mapping definitions may have references to other mapping definition.
- **F2 Additional Data.** Vertere-RDF and DataOps have no support for this feature. DataOps allows one to define predicates but subjects have to come from the dataset. As with F1, it might be complex to use this feature with some tools, for example, in OpenRefine, it is necessary to define nodes, types, predicates and values manually using a web interface.
- **F3 Select.** All analyzed tools have support for the selection of attributes from CSV datasets.
- **F4 Filter.** RML supports filters depending on the source data. For example, XPath is used for XML. For CSV data, however, RML does not support filters, as there is no such underlying technology. SML, DataOps, CSV2RDF and OpenRefine have no support for filters when applied to CSV datasets. KR2RML can be used for data cleaning and data manipulation. Hence, filters can be applied by the use of transformation functions. DataLift supports this feature using SPARQL CONSTRUCT queries. Virtuoso uploads data into a relational database in a first step. A second step allows the definition of R2RML mappings, where SQL queries with WHERE statements can be used to support this feature. RDF-Tabular and csv2rdf have partial support. For example, it is possible to apply regular expression to string types, other validation are also available for other datatypes. The tools, however, do not cover other filter options, such as comparing or combining different values from a dataset. Vertere-RDF also has partial support using regular expressions. Tarql partially supports this feature by defining a filter to skip rows. Note that all the values in the row will be skipped. A bad row is defined using SPARQL filters. For example, if an attribute has a minimum value of 10 in the RDF representation, the following filter can be applied `FILTER (?value >= 10)`. Generally, most tools that support this feature rely on other technologies, such as SPARQL CONSTRUCT queries, data transformation functions or regular expressions.
- **F5 Literal to IRI.** This feature is supported by all applications. Some tools have complex ways of defining such transformations. For example, in Datalift, you need to select an option menu. Inside this option, it is necessary to define the dataset, options for reference source data and the data to be modified. RML, as a R2RML extension, on the other hand, has a simpler approach where a template Term Map can be used to define an IRI.
- **F6 Vocabulary Reuse.** All analyzed tools have support for the reuse of RDF vocabularies and OWL ontologies.
- **F7 Transformation Functions.** As with F4, RML has support for some transformation function depending on the underlying technology used for data processing. No underlying technology is available when converting CSV data. Therefore, transformation functions are not supported for this data format. SML and Vertere-RDF have partial support for functions. User-defined functions require extending the tool. KR2RML have support for user-defined functions as Python scripts. DataOps, CSV2RDF, Tarql have no support for transformation functions. DataLift has partial support relying on SPARQL construct queries for this feature. Virtuoso relies on SQL queries for this feature as the data is imported into a relational database. Tarql also supports some transformation functions through SPARQL queries. KR2RML is the only tool with support for user-defined transformation functions as part of the mapping language.
- **F8 Datatypes.** DataOps has partial support as it is possible to define only basic XML datatypes, for example, the `xsd:dateTimeStamp` and others are not supported. All other tools have support for all XML datatypes. Together with XML datatypes, some tools allow the definition of custom datatypes, such as OpenRefine and RML.
- **F9 Named Graphs.** The specifications for RML, SML and xR2RML have support for named graphs but their implementations do not. Top Braid Composer Maestro Edition supports this feature by exporting RDF data into a specific existing named graph.

KR2RML has a triple store integrated that allows one to publish data into a specific named graph or existing ones. KR2RML is the only mapping language to support this feature.

- **F10 Blank Nodes.** DataOps is the only tool that does not support this feature. In some applications the definition of blank nodes is straightforward, such as in RML and SML. However, using DataLift or Top Braid Composer the definition of blank nodes is more complex with the use of SPARQL construct queries to refine the dataset. Only one tool has no support for this feature, as most RDF datasets would rely on blank nodes to represent complex data structures.
- **F11 Reusability.** KR2RML allows the serialization of the process as an extended R2RML mapping. In this sense, the mapping contains extra predicates with structured information, making it difficult to create or modify the mapping without their editor. Virtuoso and Top Braid Composer support this feature partially as parts of the process, such as SPIN functions for Top Braid Composer Maestro Edition, are reusable but not the whole process. DataLift and OpenRefine do not support the serialization of the uplift process. All mapping languages are reusable, as it is possible to use the same mapping file multiple times with new or updated data.

As can be seen, KR2RML is the tool with support for most features, with partial support for reusability. As mentioned before, the whole process can be serialized and reused using their editor, but several problems can be observed with the mapping. First, to reuse the mapping, you must load the data again into the editor. Second, the creation of mappings is difficult without their editor as structured information is stored as a string, requiring parsing of the mapping and the string. Finally, the data transformation functions are not reusable. It is not possible to use the same function, as functions in KR2RML do not have names nor parameters. In this sense, a function update could become difficult and prone to error – which is discussed further in Section 5.

The comparison framework we have used allows for discussion of features needed for the uplift of data into RDF. In this paper, we focus in particular on tools with support for CSV files. A key feature analyzed by our comparison framework is data transformation functions. As shown in our discussion, there is no underlying technology for CSV data. In this paper, we define a method to allow data transformation functions to be incorporated into current uplift mapping languages. The analysis and definition of approaches to deal with other problems identifiable through our comparison framework is left as future work. In the next section, a method to incorporate functions as part of uplift mapping languages called FunUL is presented. Note that in the rest of the paper Feature numbers will be included in the text (e.g. F1, F4) when talking about a particular feature.

4. FunUL – METHOD TO INCORPORATE FUNCTIONS INTO UPLIFT MAPPING LANGUAGES

FunUL is a method to incorporate data transformation by way of functions into uplift mapping languages. The method defines functions as part of the mapping in a generic, traceable, reusable and transparent way. These functions can be used to capture both domain knowledge (e.g., transforming units) and other data manipulation tasks (e.g., transforming values to create valid IRIs).

The definition of functions as part of the mapping allows data transformation and uplift into RDF to happen in a single unified step. Furthermore, functions are defined independently of the data mapping, making it possible to call the same function multiple times with different parameters. These characteristics make the uplift process into RDF more traceable, transparent and reusable. In addition, because functions are part of the mapping, it is possible to annotate them with provenance information, such as creator, creation date, and other information, such as descriptions of the transformation defined in the function and other metadata. It is also possible to discern that a certain RDF value was generated by a certain function. Moreover, functions can be shared between different mappings.

In FunUL, *functions* have a *name* and a *body*. Each function declaration must have exactly one function name and one function body. Function names are unique. Function bodies define a function using a standardized programming language. In this sense, a function body has a signature and a set of parameters. The definition of parameters is optional. Every function defined in a function body must have a return statement.

A function can perform data transformation and indirectly some data validation tasks. An example of data validation is a function defined to verify if a certain value is valid in the RDF representation. Considering that an uplift engine does not generate triples when it encounters NULL values as objects, which is the case for R2RML processors, a function that performs data validation would return a NULL value when validation fails. Section 5.1.1 shows an example of such case.

An example was already mentioned in Section 1, where a dataset may contain years using BCE/CE notation. The RDF representation, however, uses the XML datatype `xsd:gYear`. Using our method, a function would be defined to perform such transformation. The name of this function could be `yearTransformation`. The function body would have the transformation needed in a standard programming language.

The method also includes notions for calling and passing parameters to functions. A *function call* refers to a *function*. *Parameters* are optional and can be passed as references to values from the input data or as fixed values. The possibility of using parameters values allows the declaration of generic functions. For example, to call the function `yearTransformation`, there would be a reference to the function and a list of parameters, the year to be transformed (see Section 5.1.1).

The method does not rely on a specific implementation or editor. Functions in our method can perform complex data transformation, are reusable, as they can be called multiple times, and work with any data type.

4.1 FunUL Method Implementation

The FunUL implementation described in this section refers to our RML extension and to our new implementation. These implementations extend R2RML's vocabulary by introducing constructs for describing functions, function calls and parameter bindings. As mentioned before, one implementation was presented and evaluated in previous work, the RML extension [2], where functions can be applied for all input data formats supported by RML – CSV, XML, JSON and HTML. Our new implementation [3] is fully compliant with R2RML's specification, supporting F9 Named Graphs. Additionally, it has support for relational databases and CSV data. A specification of the method is also available [4].

Figure 1 shows an extended diagram with properties of term maps from R2RML and our method definitions (prefixed "rrf"). The class `rrf:Function` defines a function. A function definition has two properties defining the name, `rrf:functionName,` and the function body, `rrf:functionBody`. A function is called using the property `rrf:functionCall`. This property refers to a `rrf:Function` using the property `rr:function`. Parameters are defined using `rrf:parameterBindings`, which is an RDF Collection of term maps. Examples of function definitions and function calls can be seen in Section 5.1.1.

Figure 1

In R2RML, a Term Map generates RDF terms. An RDF term can be an IRI, a blank node or a literal. Term Maps can be values of a constant, a column or a template (see Das et al. (2012)). To implement FunUL, we define a *Function Valued* Term Map. A function call generates an RDF Term based on the function return statement defined in the function body. For example, a function can be defined to convert years in BCE/CE notation to use `xsd:gYear` XML datatype. A function call would refer to this function with a parameter value, the year. The return statement of this function would be used to generate the triples.

Analyzing the features from our comparison framework defined in Section 3, our RML extension has support for features F4 Filters and F7 Transformation Functions. F4 is supported with the use of functions. By extending RML, feature F9 Named Graphs is still not supported by this implementation. We note that RML's specification, by extending R2RML, supports named graphs. Therefore, it would be possible to implement named graphs in a RML processor. Our new implementation, an R2RML processor with support for relational databases and CSV data, does support F9 Named Graphs, as mentioned before, and therefore, supports all features defined in our comparison framework.

These implementations load functions using Java's Nashorn [5] JavaScript engine available in the `javax.script` package. JavaScript was chosen (even though functions in our method could be defined in any programming language) because it is freely available, widely used and its specification is an ISO standard. Any errors loading or executing the function are reported back to the user. Currently there is no support for monitoring functions, relying on Nashorn and the Java Runtime Environment to handle any problems related to memory management and correctness of the code.

5. EVALUATION

In this section, we compare our method with that of KR2RML using two real world use cases, the Seshat Use Case and the Ordnance Survey Ireland Use Case. This comparison evaluated how well both approaches could implement the required mappings. In this evaluation, we use our new implementation.

5.1 The Seshat Use Case

The first use case comes from a project called Seshat: Global History Databank (Turchin et al., 2015). This international project led by the Evolution Institute (USA) and the University of Oxford is developing a knowledge base to describe human societies over the last 15,000 years as a set of time series. This knowledge base is structured according to a social sciences "codebook" or schema specified by an editorial board of domain experts in structured natural language and re-engineered into an OWL ontology by knowledge engineers at Trinity College Dublin (Brennan et al., 2016). The codebook specifies over 1,000 data variables of interest cover topics such as social complexity measures, warfare, technology, ritual and so on. Two main units of data collection and analysis are specified – the Polity (society) and the Natural Geographical Area (NGA) but 22 distinct units of collection are currently used and this number continues to expand as the project matures. Each variable is not modeled as a simple value or object instance and is subject to uncertainty, temporal and geographical bounds for its validity. All of this must be explicitly modeled in the final OWL representation.

The initial data collection effort (2011 – 2016) used a wiki structured according to the codebook. The natural language codebook or a sub-set was used as a template for each wiki page to be completed describing a single unit of analysis, typically a temporally bounded Polity (human society). Within Seshat there is a hierarchical distribution of effort between teams of research assistants (typically about 10-15 active at any one time in 3-5 data collection locations) who manually research and enter data, Seshat researchers who evolve the codebook and direct the data collection effort to particular geo-temporal entities based (typically 20 qualified to at least PhD level) and over 60 external domain experts who validate data (drawn from the worldwide pool of domain experts, typically full professors). At present the dataset contains over 120,000 expert-curated "facts" but each one of these is qualified in terms of uncertainty, disagreement, academic sources so that it may require 100 triples to describe it fully. The current collection effort is focused on an initial 30 NGAs distributed across the globe to maximize the distribution of societies examined. These facts form time series at a sample rate of 100 years that describe all human societies in the 30 NGAs from approximately 10,000 BC to the industrial revolution. The current wiki MySQL DB is over 4GB, with 1081 pages describing units of collection such as Polities and NGAs. Each page typically has over 1000 variables describing the Polity.

To facilitate data collection and analysis the pages use structured natural language with a well-defined syntax for describing variables, values, uncertainty, temporal bounds and annotations. In May 2014, Trinity College Dublin developed a web scraper tool that is aware of this syntax and can either validate a page to detect syntactic errors (for use by the RAs during data entry) or dump the page as a TSV file. A bulk export mode is also available whereby the entire wiki or scoped sub-sets can be dumped into a TSV file. The TSV files are then used by statisticians to model human societies based on the data in the wiki. Although not designed with this purpose in mind, these TSV files can provide a starting point for uplifting the wiki into RDF based on the new Seshat OWL Ontology.

5.1.1 Our approach

One of the issues in the uplift of the Seshat dataset into RDF is that predicates in the data differ from the predicates defined in the OWL ontology. Another issue was already mentioned in Section 1. Years in the dataset follow a BCE/CE notation, but the ontology uses the XML datatype `xsd:gYear`. Another example of transformation would be the use of a split function. In the dataset, some values are stored in one attribute, but the ontology defines different predicates for each part of the value. **Listing 1** shows a fragment of the dataset.

```
NGA,Polity,Variable,Value From,Value To,Date From,Date To
Latium,ItRomPr,RA,Edward A L Turner,,,
Latium,ItRomPr,Expert,Garrett Fagan,,,
Latium,ItRomPr,Peak Date,117 CE,,,
Latium,ItRomPr,Duration,31 BCE - 284 CE,,,
Latium,ItRomPr,Polity territory,4500000,,14CE,
```

Listing 1: Fragment of the Seshat dataset

Transformation functions can be defined to overcome these issues. “RA” and “Expert” have specific predicates so it is possible to use a function to evaluate if it the triple should be generated. The function to do so, using our method, is defined in **Listing 2**. This function performs a data validation task, returning NULL when the triple should not be generated. For example, a mapping to generate the predicate `seshat:ra` for the Seshat dataset, would call this function with parameters `rr:column "Variable"`, `rr:constant "RA"` and `rr:column "Value From"`. In this sense, the triple will be generated only when the attribute “Variable” has value “RA”.

```
<#Check>
 rrf:functionName "check" ;
 rrf:functionBody """
     function check(var1, var2, value) {
       if(var1 == var2) {
          return value;
       }
       return null;
     }
""" ; .
```

Listing 2: Function to check if a value should be generated

For the attribute “Peak Date”, the value needs to be transformed to use the XML datatype `xsd:gYear`. This function is shown in **Listing 3**. As functions are resources in the same RDF file, it is possible to reuse it many times. Note that the mapping will define the datatype `xsd:gYear` - as it is shown in **Listing 6** with the use of the predicate `rr:datatype`.

```
<#YearTransformation>
 rrf:functionName "yearTransformation" ;
 rrf:functionBody """
     function yearTransformation (year) {
        year = year.trim();
        if(year.indexOf("BCE") > -1){
            return String(parseInt("-" + year.replace("BCE", "")) + 1);
        }
        return year.replace("CE", "").trim();
     }
""" ; .
```

Listing 3: Function to transform the year

For the attribute “Duration”, one would need to split the value first and then apply the data transformation. As it is shown in **Listing 4**, in our implementation, it is possible to call other functions inside a function. This function only applies the year transformation function for a specific attribute, in this case “Duration”. For the last line of the dataset showed in **Listing 1**, we reuse the function used for the attribute “Peak Date”. Note that this function deals with blank spaces as well – in the dataset, not all year values are separated by spaces. For example, we have the value “117 CE” with a space, and then “14CE”.

```
<#SplitAndYearTransformation>
 rrf:functionName "splitAndYearTransformation" ;
 rrf:functionBody """
    function
       split(variable, value, check, index, separator) {
          if(variable == check) {
          var str = value.split(separator)[index].trim();
          return yearTransformation(str);
        }
        return null;
     }
""" ; .
```

Listing 4: Function to split a value and transform the year

Listing 5 shows how to call a function using our method. This function has five parameters, two parameters come from the dataset using `rr:column`, the others are constants, `rr:constant`. In this sense, the function called is generic and could be reused for other parameters.

```
<#DurationBeginning>
 rr:logicalTable [
    rr:tableName "data" ;
 ] ;

 rr:subjectMap [
    rr:termType rr:BlankNode;
    rr:class owltime:DateTimeDescription
 ];

 rr:predicateObjectMap [
    rr:predicate owltime:year;
    rr:objectMap [
      rr:termType rr:Literal;
      rr:datatype xsd:gYear;
      rrf:functionCall [
        rrf:function <#SplitAndYearTransformation> ;
        rrf:parameterBindings (
          [ rr:column "Variable" ]
          [ rr:column "Value From" ]
          [ rr:constant "Duration" ]
          [ rr:constant "0" ]
          [ rr:constant "-" ]
       ) ;
      ] ;
    ];
 ].
```

Listing 5: Mapping with a function call to `yearTransformation`

The output of the uplift process using these functions applied to the dataset showed in **Listing 1** can be in **Listing 6**.

```
@base         <http://dacura.cs.tcd.ie/data/seshat> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix time: <http://www.w3.org/2006/time#> .
<seshat/ItRomPr> <#RA> "Edward A L Turner" .
<seshat/ItRomPr> <#Expert> "Garrett Fagan" .
<seshat/ItRomPr> <#peakDate> _:lLA98YAitY .
_:lLA98YAitY a <#TemporalInstantVariable> .
_:lLA98YAitY <#definiteValue> _:TERpMusPG9 .
_:TERpMusPG9 a <#Instant> .
_:TERpMusPG9 <#atDateTime> _:P3qgpVj36x .
_:P3qgpVj36x a time:DateTimeDescription .
_:P3qgpVj36x time:year "117"^^xsd:gYear .
_:P3qgpVj36x time:unitType time:unitYear .
<seshat/ItRomPr> <#duration> _:awMl8Sww0N .
_:awMl8Sww0N a <#DurationVariable> .
_:awMl8Sww0N <#definiteValue> _:HzsjbPE9RU .
_:HzsjbPE9RU a <#Interval> .
_:HzsjbPE9RU <#hasBeginning> _:wUMzVkWduq .
_:wUMzVkWduq a time:DateTimeDescription .
_:wUMzVkWduq time:unitType time:unitYear .
_:wUMzVkWduq time:year "-30"^^xsd:gYear .
_:HzsjbPE9RU <#hasEnd> _:B1hn2AyEdl .
_:B1hn2AyEdl a time:DateTimeDescription .
_:B1hn2AyEdl time:year "284"^^xsd:gYear .
_:B1hn2AyEdl time:unitType time:unitYear .
<seshat/ItRomPr> <#territory> _:IL9hDo2Izd .
_:IL9hDo2Izd a <#TerritoryVariable> .
_:IL9hDo2Izd a <#Instant> .
_:IL9hDo2Izd <#definiteValue> "4500000"^^xsd:unsignedLong .
_:IL9hDo2Izd <#atDateTime> _:kzsZr2yrBX .
_:kzsZr2yrBX a time:DateTimeDescription .
_:kzsZr2yrBX time:unitType time:unitYear .
_:kzsZr2yrBX time:year "14"^^xsd:gYear .
```

Listing 6: Seshat's use case RDF output

5.1.2 KR2RML's approach

For this comparison, we will use KR2RML's editor to define a function similar to the one presented in **Listing 2**. Functions in KR2RML are defined in Python. This function generates a specific predicate when the attribute "Variable" has the value "RA". The function exported as a KR2RML mapping is shown in **Listing 7**. One can see that – next to the RDF file – structured information is contained as a literal in the file. In KR2RML, three things need to be parsed: the RDF file, the structured information in the literal, and finally the functions in Python.

```
@prefix km-dev: <http://isi.edu/integration/karma/dev#> .
_:node1afgfa0n8x1 a km-dev:R2RMLMapping ;
  ...
  km-dev:hasWorksheetHistory """[{
  ...
  {
    \"name\": \"transformationCode\",
    \"type\": \"other\",
    \"value\": \"return getValue(\\\"Value From\\\") if getValue(\\\"Variable\\\") == \\\"RA\\\" else \\\"\\\"\"
  },
  ...
}]""" .
```

Listing 7: Function check in a KR2RML mapping

5.1.3 Discussion

In both approaches data transformation functions can be defined within mapping definitions, but KR2RML's functions are not reusable. It is possible to reapply a function by accessing all used functions using the editor, but it is not possible to call the same function multiple times. In this sense, a function needs to be implemented for every possible parameter value. In contrast, functions in our method can be reused many times with different parameters. More specifically, for example, in our method we call the function defined in **Listing 2** twice, with different parameters. Firstly, with the constant parameter "RA" to create a specific predicate. The same function is called a second time to define the predicate for the value "Expert". In KR2RML, another function, similar to the one defined in **Listing 7**, needs to be defined for the second case, changing the value "RA" to "Expert". As mentioned before, this characteristic makes function updates complex and prone to error. Other problems with functions in KR2RML include, as can be seen in **Listing 7**, the definition of other structured information together with functions as strings. This requires the mapping file to be parsed three times. Furthermore, the mapping file becomes complex and the mapping language heavily dependent on its editor.

Another issue encountered when creating mappings for the Seshat dataset using KR2RML is the use of functions to create predicates. KR2RML only allows functions to be applied to input data. Classes and properties in KR2RML are defined from ontologies that need to be imported into the editor. In this sense, to define predicates from the dataset, it is necessary to import the RDF vocabulary. This vocabulary allows one to use RDF Reification. In this sense, one can define a node of type `rdf:Statement`, and then use the properties `rdf:subject`, `rdf:predicate` and `rdf:object` to define triples. However, the use of this vocabulary increases mapping complexity. In contrast, our approach allows the use of functions to create subjects, predicates and objects directly.

5.2 The Ordnance Survey Ireland (OSi) Use Case

Our second use case was provided by Ireland's national mapping agency, the Ordnance Survey Ireland (OSi) [6]. In 2014, OSi delivered a newly developed spatial data storage model known as Prime2 (Prime2, 2014). With Prime2, OSi moved from a traditional map-centric model towards an object-oriented model from which various types of mapping and data services can be produced. Prime2 and the associated workflows furthermore incorporated governance practices to cope with evolution of spatial objects in their model. The system currently holds information of over 45,000,000 spatial objects (road segments, buildings, fences, etc.), of which some have more than one representation. These objects are stored in an Oracle Spatial and Graph database [7].

The OSi furthermore aims to leverage user engagement with their geospatial information (and derived maps), which has a legal weight in Ireland. One of the initiatives they launched is called GeoHive [8], allowing one easy access to publically available spatial data. An ongoing project has made the OSi data available as Linked Data, which requires the uplift of this data into RDF (see Debruyne et al. (2016) for more information about the OSi Linked Data Platform).

5.2.1 Our approach

One of the use cases provided by the OSi is in relation to centroids. In the dataset, geometries are represented as polygons using the Well-Known Text (WKT) markup language. As a requirement, in the RDF representation, together with polygons from the dataset, the centroid of these must be expressed. Centroids can give a better clue of the size or location of a building with respect to streets, for example. **Listing 8** shows a fragment of the OSi dataset.

```
GUID,FIRST_COUNTY,FIRST_CONTAE,FIRST_PROVINCE,FIRST_TD_ENGLISH,FIRST_TD_GAEILGE,GEOM
"2AE1962A07E013A3E055000000000001","DUBLIN","Baile Átha Cliath","Leinster","CORK GREAT","POLYGON ((-6.1172386044796
53.2106905753062, -6.11424909966871 53.207979467197, -6.10861907330405 53.2111812425034, -6.10377172584186
53.2108334615519, -6.10277870491329 53.2107728977641, -6.10501756498507 53.2184440038722, -6.1195203311062
53.2159055411092, -6.1172386044796 53.2106905753062))"
"2AE1962A07E113A3E055000000000001","DUBLIN","Baile Átha Cliath","Leinster","CORK LITTLE","POLYGON ((-6.12181623793855
53.221888252828, -6.1195203311062 53.2159055411092, -6.10501756498507 53.2184440038722, -6.10762054241042
53.2273511279793, -6.11036323642191 53.2257671814042, -6.10862377470503 53.2237584373442, -6.11796597954464
53.2224646519635, -6.12181623793855 53.221888252828))"
"2AE1962A07E713A3E055000000000001","DUBLIN","Baile Átha Cliath","Leinster","CORRAGEEN","POLYGON ((-6.36241459852623
53.2515671670277, -6.36013382263472 53.2500325570601, -6.35795924791704 53.2521491523323, -6.36119357565768
53.2534862178303, -6.36053907014501 53.2553230424802, -6.36241459852623 53.2515671670277))"
```

Listing 9: Fragment of the OSi dataset

Many libraries are available with support for the calculation of centroids from polygons. In this use case, we have used two libraries: Turf [9], a JavaScript library for spatial analysis; and Terraformer [10], another JavaScript library geo toolkit. Turf is used to calculate the centroid of polygons and Terraformer is used to parse objects from WKT format into GeoJSON and vice versa. Turf's library uses GeoJSON in the analysis and manipulation of geospatial data. The RDF output uses the WKT format. We have modified the Terraformer library so that it could be used and referred to within the R2RML processor JavaScript engine. This modification added a global variable instantiating a Terraformer object. The Turf library was not modified. The function definition using our approach is shown below. The first three lines of the function body load the aforementioned libraries into R2RML's JavaScript engine.

```
<#Centroid>
 rrf:functionName "centroid" ;
 rrf:functionBody """
     load("turf.min.js");
     load("terraformer.js");
     load("terraformer-wkt-parser.js");
     function centroid(polygon) {
       var centroid = turf.centroid(Terraformer.WKT.parse(polygon));
       return Terraformer.WKT.convert(centroid.geometry);
     }
""" ; .
```

Listing 10: Function definition for calculating centroids

A function call to the function defined in **Listing 10** is shown in **Listing 11**. In this mapping the original value of the attribute GEOM is also used to create another triple. This object of this triple has the datatype `geo:wktLiteral` and subjects have the type class `geo:Geometry` from GeoSPARQL (Perry and Herring, 2012). GeoSPARQL defines a vocabulary for representing geospatial data in RDF, and an extension of the SPARQL query language for processing geospatial data.

```
<#TMCentroid>
  rr:logicalTable [
    rr:tableName "data" ;
  ] ;

  rr:subjectMap [
    rr:class geo:Geometry ;
    rr:termType rr:BlankNode ;
    rr:column "GUID" ;
  ] ;

  rr:predicateObjectMap [
    rr:predicate geo:asWKT ;
    rr:objectMap [
      rr:column "GEOM" ;
      rr:datatype geo:wktLiteral ;
    ] ;
  ] ;

  rr:predicateObjectMap [
    rr:predicate geo:point ;
    rr:objectMap [
      rrf:functionCall [
        rrf:function <#Centroid> ;
        rrf:parameterBindings (
          [ rr:column "GEOM" ]
        ) ;
      ] ;
    ] ;
  ] ;.
```

Listing 11: Mapping with a function call to `centroid`

The RDF output of the mapping shown in **Listing 11** using the data from **Listing 12** is shown below.

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
[] a geo:Geometry ;
  geo:asWKT "POLYGON ((-6.1172386044796 53.2106905753062, -6.11424909966871 53.207979467197, -6.10861907330405
53.2111812425034, -6.10377172584186 53.2108334615519, -6.10277870491329 53.2107728977641, -6.10501756498507
53.2184440038722, -6.1195203311062 53.2159055411092, -6.1172386044796 53.2106905753062))"^^geo:wktLiteral ;
  geo:point "POINT (-6.1101707291855405 53.21225816990057)".

[] a geo:Geometry ;
  geo:asWKT "POLYGON ((-6.12181623793855 53.2218888252828, -6.1195203311062 53.2159055411092, -6.10501756498507
53.2184440038722, -6.10762054241042 53.2273511279793, -6.11036323642191 53.2257671814042, -6.10862377470503
53.2237584373442, -6.11796597954464 53.2224646519635, -6.12181623793855 53.2218888252828))"^^geo:wktLiteral ;
  geo:point "POINT (-6.11298966673026 53.22222568127934)".

[] a geo:Geometry ;
  geo:asWKT "POLYGON ((-6.36241459852623 53.2515671670277, -6.36013382263472 53.2500325570601, -6.35795924791704
53.2521491523323, -6.36119357565768 53.2534862178303, -6.36053907014501 53.2553230424802, -6.36241459852623
53.2515671670277))"^^geo:wktLiteral;
  geo:point "POINT (-6.360448062976135 53.252511627346124)".
```

Listing 13: OSi's use case RDF output

5.2.2 KR2RML's approach

We have applied the same approach to KR2RML, relying on the Python package Shapely [11] to calculate centroids from polygons. Shapely is a library for manipulation and analysis of planar geometric objects based on GEOS [12]. KR2RML's engine uses Jython [13], a Java implementation of the Python programming language, to execute data transformation functions. However, the Jython implementation used in KR2RML has no support for some functions used by the Shapely package. In this regard, we have defined a python script that uses Shapely to calculate centroids and a python function within KR2RML that calls this script. **Listing 14** shows the KR2RML mapping with the Python transformation function to execute the script shown in **Listing 15**.

```
@prefix km-dev: <http://isi.edu/integration/karma/dev#> .

_:node1bb6618d0x1 a km-dev:R2RMLMapping ;
    ...
    km-dev:hasWorksheetHistory """[ {
```

```
    ...
    {
       \"name\": \"transformationCode\",
       \"type\": \"other\",
       \"value\":    \"import    commands\\nreturn    commands.getoutput('python    shapely-function.py    \\\\\\"'    +
getValue(\\\"GEOM\\\") + '\\\\\\"')\\n\"
            },
    ...
    }
]""" .
```

Listing 14: Function to call a script that calculates centroids in a KR2RML mapping

```
import sys
from shapely.wkt import loads as load_wkt
print load_wkt(sys.argv[1]).centroid.wkt
```

Listing 15: Using the shapely library to calculate centroids

5.2.3 Discussion

Both approaches can be used to calculate centroids from polygons. However, as mentioned before, functions in KR2RML have no signatures or parameters and therefore, they are not reusable. This requires the definition of a new function for every time centroids need to be calculated. In contrast, our approach defines function as resources that can be referred to.

Other issues, not related to the use of functions, were found when using the KR2RML's editor. One issue is the definition of datatypes. The OSi dataset requires specific datatypes from geospatial vocabularies, such as `geo:wktLiteral`, as can be seen in the R2RML mapping shown in **Listing 11** and, therefore, in the RDF output shown in **Listing 13**. KR2RML supports different datatypes - as an R2RML extension - but its editor does not. The editor only allows the definition of XML datatypes. As mentioned before, the serialization of KR2RML mappings uses strings to define structure information together with mapping definitions, what makes the mapping file complex and the creation and editing of mappings without the editor troublesome. Another problem is the definition of blank nodes. First, all blank nodes defined using the editor need a class type, which is not always the case when data is being uplifted. Second, when defining a certain number of blank nodes, the editor would give an error saying that the serialization of the mapping was not possible. A final remark is about the graph visualization, where mapping complex RDF structures are difficult because of the data centric approach, where a graph representing the mapping definition is shown together with the source data represented as a table.

6. CONCLUSION

In this paper, a comparison framework to evaluate uplift tools applied to CSV datasets was presented. Relying on one of the features – transformation functions – evaluated by this framework, we described FunUL. FunUL is a method to incorporate functions into uplift mapping languages. The general approach for data transformation during the uplift process of CSV data into RDF relies on converting the source data into another format or on pre/post-processing techniques. In contrast, functions in our method are defined as part of the mapping, integrating transformation functions and mapping definitions. This makes the uplift process, in regard to data transformations, transparent and traceable. The evaluation applied the method to two real world use cases and compared the use of functions as part of the mapping to KR2RML, the only other uplift tool identified to have this feature. This evaluation showed that even though the whole process can be serialized using KR2RML, their functions are not reusable, which can make function updates complex and prone to error. Furthermore, KR2RML's mappings are stored as strings, which makes the mapping file complex (i.e., parsing the RDF file and parsing the strings that relate fields to functions), relying heavily on their editor and making the creation and editing of mappings difficult using other tools. Finally, KR2RML only allows the use of functions to the input data. In contrast, FunUL define functions as resources that can be used multiple times, with different parameters – what facilitates function updates – and it does not rely on a specific editor. Additionally, functions can be applied in the creation of subjects, predicates and objects. Other problems with the KR2RML editor, not related to function definitions, were also identified for the presented use cases. Examples of such problems are the definition of blank nodes and datatypes.

Future work includes extending the method to better describe functions. One example of an ontology developed to semantically declare and describe functions is presented in Meester et al. (2016), called the Function ontology. R2RML-F can fetch functions published on the Web using Linked Data principles, which indicates that it would be possible to do the same with functions declared using the Function ontology or other technologies. Future work also includes the use of other programing languages to define functions as part of the mapping and additional experiments and use cases, such as the use of functions to generate provenance information during the uplift process (see Dimou et al. (2016)). A limitation of having function definitions within the mapping is the domain knowledge required from users in the definition of functions within mappings.

Notes

1. `https://github.com/antidot/db2triples`
2. `https://github.com/CNGL-repo/RMLProcessor(Accessed March 2017)`
3. `https://opengogs.adaptcentre.ie/debruync/r2rml(Accessed March 2017)`
4. `https://www.scss.tcd.ie/~crottija/funul/(Accessed March 2017)`

5. https://blogs.oracle.com/nashorn/(Accessed March 2017)

6. https://www.osi.ie (Accessed March 2017)

7. https://www.oracle.com/database/spatial/(Accessed March 2017)

8. http://www.geohive.ie(Accessed March 2017)

9. http://turfjs.org/(Accessed March 2017)

10. http://terraformer.io/(Accessed March 2017)

11. https://github.com/Toblerity/Shapely (Accessed March 2017)

12. https://trac.osgeo.org/geos/(Accessed March 2017)

13. http://www.jython.org/(Accessed March 2017)

REFERENCES

Biron, P., and Malhotra, A. (2004). "XML Schema Part 2: Datatypes Second Edition", available at https://www.w3.org/TR/xmlschema-2/.

Bizer, C. and Seaborne, A. (2004), "D2RQ - Treating non-RDF Databases as Virtual RDF Graphs", in *Proceedings of the 3rd international semantic web conference (ISWC 2004)*.

Brennan, R., Feeney, K., Mendel-Gleason, G., Bozic, B., Turchin, P., Whitehouse, H., Francois, P., Currie, T. E. and Grohmann, S. (2016), "Building the Seshat Ontology for a Global History Databank", in *The Extended Semantic Web Conference, ESWC 2016*, pp. 693–708.

Das, S., Sundara, S. and Cyganiak, R. (2012) "R2RML: RDB to RDF Mapping Language", available at https://www.w3.org/TR/r2rml/.

Debruyne, C. and O'Sullivan, D. (2016), "R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings", in *Proceedings of the Workshop on Linked Data on the Web, LDOW 2016, co-located with the 25th International World Wide Web Conference (WWW 2016)*.

Debruyne, C., Clinton, E., McNerney, L., Nautiyal, A. and O'Sullivan, D. (2016), "Serving Ireland's Geospatial Information as Linked Data", in *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016)*.

Dimou, A., De Nies, T., Verborgh, R., Mannens, E., and Van de Walle, R. (2016), "Automated Metadata Generation for Linked Data Generation and Publishing Workflows", in *Proceedings of the Workshop on Linked Data on the Web, LDOW 2016, co-located with the 25th International World Wide Web Conference (WWW 2016)*.

Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E. and Van de Walle, R. (2014), "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data", in *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014)*.

Harris, S., Seaborne, A. and Prud'hommeaux, E. (2013). "SPARQL 1.1 Query Language", available at https://www.w3.org/TR/sparql11-query.

Hert, M., Reif, G. and Gall, H.C. (2011), "A comparison of RDB-to-RDF Mapping Languages", in *Proceedings of the 7th International Conference on Semantic Systems. I-Semantics*, ACM, pp. 25-32.

Heyvaert, P., Dimou, A., Herregodts, A.L., Verborgh, R., Schuurman, D., Mannens, E. and Van de Walle, R. (2016), "RMLEditor: A Graph-based Mapping Editor for Linked Data Mappings", in *The Extended Semantic Web Conference, ESWC 2016*, pp. 709-723.

Hitzler, P., Krotzsch, M. and Rudolph, S. (2009), "Foundations of Semantic Web Technologies". CRC Press.

Junior, A. C., Debruyne, C. and O'Sullivan, D. (2016) "Incorporating Functions in Mappings to Facilitate the Uplift of CSV Files into RDF", in *The Semantic Web - ESWC 2016 Satellite Events*, pp. 55–59.

Junior, A. C., Debruyne, C., Brennan, R. and O'Sullivan, D. (2016), "FunUL: a Method to Incorporate Functions into Uplift Mapping Languages", in *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services, iiWAS 2016*, pp. 267–275.

Klyne, G., Jeremy J. J. and McBrid, B., (2004), "Resource description framework (RDF): Concepts and abstract syntax", available at https://www.w3.org/TR/rdf11-concepts/.

Knublauch, H., Hendler, J.A. and Idehen, K. (2009). "SPIN – SPARQL Inferencing Notation", available at http://spinrdf.org/.

Michel, F., Djimenou, L., Faron-Zucker, C. and Montagnat, J. (2015), "Translation of relational and non-relational databases into RDF with xR2RML", in *11th Web Information Systems and Technologies (WEBIST 2015)*, pp 443-454.

Meester, B.D., Dimou, A., Verborgh, R. and Mannens, E. (2016), "An Ontology to Semantically Declare and Describe Functions", in *The Semantic Web - ESWC 2016 Satellite Events*, pp. 46–49.

Perry, M. and Herring, J. (2012), "GeoSPARQL-A Geographic Query Language for RDF data", available at http://www.opengeospatial.org/standards/geosparql.

Pinkel, C., Schwarte, A., Trame, J., Nikolov, A., Bastinos, A.S. and Zeuch, T. (2015), "DataOps: Seamless End-to-End Anything-to-RDF Data Integration", in *The Semantic Web - ESWC 2015 Satellite Events*, pp 123-127.

"Prime2: Data Concepts and Data Model Overview". Tech. rep., Ordnance Survey Ireland (2014), available at http://www.osi.ie/wp-content/uploads/2015/04/Prime2-V-2.pdf (accessed March 2017).

Pollock, R. Tennison, J., Kellogg, G. and Herman, I. (2015), "Metadata Vocabulary for Tabular Data", available at https://www.w3.org/TR/tabular-metadata/.

Purohit, S., Smith, W., Chappell, A., West, P., Lee, B., Stephan, E. and Fox, P. (2016), "Effective Tooling for Linked Data Publishing in Scientific Research", in *2016 IEEE Tenth International Conference on Semantic Computing (ICSC 2016),* pp 24-31.

Scharffe, F., Atemezing, G., Troncy, F., Gandon, F., Villata, S., Bucher, B., Hamdi, F., Bihanic, L., Képéklian, G., Cotton, F., et al. (2012), "Enabling linked data publication with the Datalift platform", in *Proc. AAAI Workshop on Semantic Cities*.

Slepicka, J., Yin, C., Szekely, P. and Knoblock, C. (2015), "KR2RML: An alternative interpretation of R2RML for heterogeneous sources", in *Proceedings of the 6th International Workshop on Consuming Linked Data*.

Stadler, C., Unbehauen, J., Westphal, P., Sherif, M.A. and Lehmann, J. (2015), "Simplified RDB2RDF Mapping", in *Proceedings of the Workshop on Linked Data on the Web co-located with the 24rd International World Wide Web Conference (WWW 2015)*.

Tennison, J., Kellogg, G. and Herman, I. (2015), "Model for Tabular Data and Metadata on the Web", available at https://www.w3.org/TR/tabular-data-model/.

Turchin, P., Brennan, R., Currie, T., Feeney, K., Francois, P., Hoyer, D., Manning, J., Marciniak, A., Mullins, D., Palmisano, A., et al. (2015), "Seshat: The global history data-bank", in *Cliodynamics: The Journal of Quantitative History and Cultural Evolution 6*.

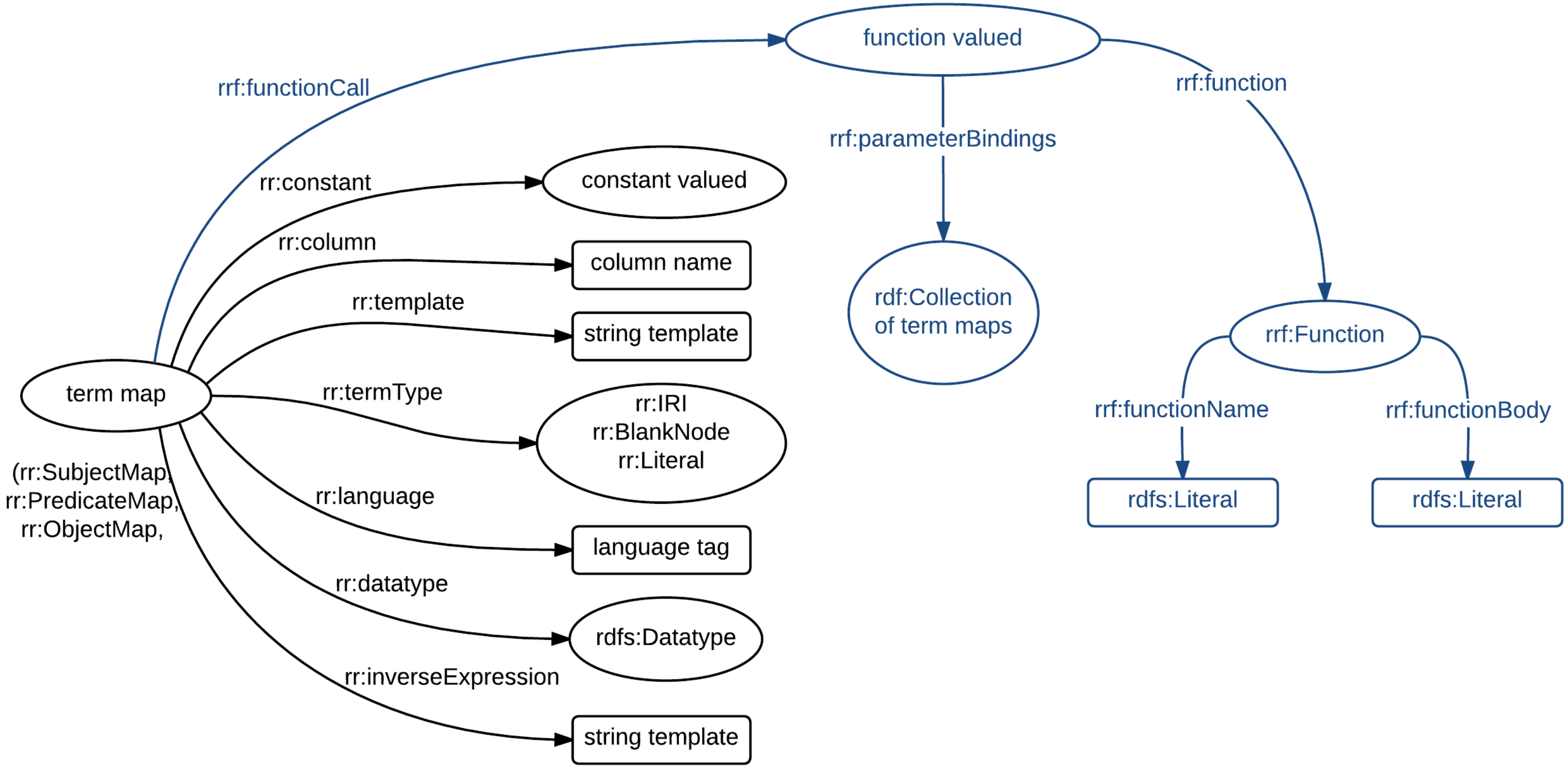
rrf:functionCall
function valued
rrf:function
rrf:parameterBindings
rdf:Collection of term maps
rrf:Function
rrf:functionName
rrf:functionBody
rdfs:Literal
rdfs:Literal
rr:constant
constant valued
rr:column
column name
rr:template
string template
term map
rr:termType
rr:IRI
rr:BlankNode
rr:Literal
(rr:SubjectMap,
rr:PredicateMap,
rr:ObjectMap,
rr:language
language tag
rr:datatype
rdfs:Datatype
rr:inverseExpression
string template

Table 1: State-of-the-Art in CSV Uplift.

Description	Additional Software Support	Mapping Language
DataLift (Scharffe et al., 2012) is a modular open source platform where one needs to convert data into raw RDF as a first step. After that, it is possible to explore and transform the RDF representation.	✓	
The Virtuoso Universal Server [1] has an extension where CSV datasets can be imported into relational databases. After loading the data, mapping into RDF can be done by defining R2RML mappings manually or by using a wizard that generates R2RML mappings.	✓	✓
TopBraid Composer Maestro Edition [2] converts data into RDF with no customizations in a first step. A second step is used to transform the RDF representation using SPARQL (Harris et al., 2013) construct queries and SPIN (Knublauch et al., 2009) rules.	✓	
DataOps (Pinkel et al., 2015**)** is described as a semantic Anything-to-RDF Extract Transform Load (ETL) data integration tool. DataOps works in three main steps. The first step allows one to access data in different formats. The second allows the mapping into RDF, where data can be manipulated. Finally, the tool offers options for interlinking the RDF dataset with other existing datasets.	✓	
OpenRefine [3] has support for cleaning and transformation functions for many data formats. The uplift to RDF is available through RDF Refine [4], an extension to OpenRefine, allowing the mapping and interlinking of RDF datasets through a web interface.	✓	
KR2RML (Slepicka et al., 2015) is an extension to the W3C recommendation R2RML. KR2RML is developed on top of Karma, a data integration tool with support for many data formats as input. The tool also supports data cleaning and data transformation.	✓	✓
RML (Dimou et al., 2014) is defined as a superset of R2RML, a W3C recommendation for mapping relational databases into RDF. RML extends R2RML's vocabulary to support a broader set of possible input data formats, including CSV and others. RML also has an additional software tool that uses a graph representation to define RML mappings called RMLEditor (Heyvaert et al., 2016).	✓	✓
xR2RML (Michel et al., 2015), following RML, extends R2RML's vocabulary to support the uplift of various data formats into RDF. The language also supports RDF collections, which are not supported by R2RML.		✓
SML (Stadler et al., 2015) is defined as a human readable syntax mapping language that currently supports relational databases and CSV datasets. SML is based on SQL CREATE VIEWS and SPARQL construct queries.		✓
CSV2RDF [5] applies the use of a template file to determine how one row of a CSV dataset will be represented in RDF. The template file follows the RDF Turtle like syntax.		✓
RDF-Tabular [6] is an implementation of the CSVW specifications, a W3C recommendation for dealing with CSV datasets. In summary, the W3C CSWV Working group defined a Model for Tabular Data (Tennison et al., 2015) as an abstract model. This model also allows the use of metadata through the Metadata Vocabulary for Tabular Data (Pollock et al., 2015). The metadata provides one ways of annotating tabular data, defining the structure of this data. Together, these specifications define ways to process tabular data for many purposes, such as conversions, validation and others.		✓
CSV 2 RDF [7**]** is another fully compliant implementation of the CSVW specifications for the uplift of CSV data into RDF.		✓
Tarql [8] is a mapping language based on SPARQL CONSTRUCT queries to convert CSV datasets into RDF. The language uses SPARQL query language for its mappings.		✓
Vertere-RDF [9] is another mapping language that uses a template mechanism to define mappings. The mapping file follows the RDF Turtle syntax to express how CSV data should be represented in RDF, being similar to CSV2RDF.		✓

[1] http://virtuoso.openlinksw.com/(Accessed March 2017)

2. http://www.topquadrant.com/(Accessed March 2017)

3. http://openrefine.org/(Accessed March 2017)

4. http://refine.deri.ie(Accessed March 2017)

5. https://github.com/clarkparsia/csv2rdf(Accessed March 2017)

6. https://github.com/ruby-rdf/rdf-tabular(Accessed March 2017)

7. https://github.com/theodi/csv2rdf(Accessed March 2017)

8. https://github.com/tarql/tarql(Accessed March 2017)

9. https://github.com/mmmmmrob/Vertere-RDF(Accessed March 2017)

Table 2: Comparison framework applied to the state-of-the-art in CSV Uplift.

Tools	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
DataLift	✓	✓	✓	✓	✓	✓	(✓)	✓	✓	✓	✖
Virtuoso	✓	✓	✓	✓	✓	✓	(✓)	✓	✓	✓	(✓)
TopBraid Composer Maestro Edition	✓	✓	✓	✓	✓	✓	✓	✓	(✓)	✓	(✓)
DataOps	✖	✖	✓	✖	✓	✓	✖	(✓)	✖	✖	✖
OpenRefine	✓	✓	✓	✖	✓	✓	✓	✓	✖	✓	✖
KR2RML	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	(✓)
RML	✓	✓	✓	✖	✓	✓	✖	✓	✖	✓	✓
SML	✓	✓	✓	✖	✓	✓	✖	✓	✖	✓	✓
CSV2RDF	✓	✓	✓	✖	✓	✓	✖	✓	✖	✓	✓
RDF-Tabular	✓	✓	✓	(✓)	✓	✓	(✓)	✓	✖	✓	✓
CSV 2 RDF	✓	✓	✓	(✓)	✓	✓	(✓)	✓	✖	✓	✓
Tarql	✓	✓	✓	(✓)	✓	✓	(✓)	✓	✖	✓	✓
Vertere-RDF	✓	✖	✓	✖	✓	✓	(✓)	✓	✖	✓	✓

✓ = full support (✓) = partial support ✖ = no support