Ademar Crotti Junior, Christophe Debruyne, Rob Brennan, and Declan O'Sullivan. Funul: a method to incorporate functions into uplift mapping languages. In Gabriele Anderst-Kotsis, editor, Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services, iiWAS 2016, Singapore, November 28-30, 2016, pages 267–275. ACM, 2016

FunUL: A Method to Incorporate Functions into Uplift Mapping Languages

Ademar Crotti Junior
Trinity College Dublin
College Green
Dublin 2
crottija@scss.tcd.ie

Christophe Debruyne
ADAPT Centre
Trinity College Dublin
College Green
Dublin 2
debruync@scss.tcd.ie

Rob Brennan
ADAPT Centre
Trinity College Dublin
College Green
Dublin 2
rob.brennan@scss.tcd.ie

Declan O'Sullivan
ADAPT Centre
Trinity College Dublin
College Green
Dublin 2
declan.osullivan@scss.tcd.ie

ABSTRACT

Typically tools that map non-RDF data into RDF format rely on the technology native to the source of the data when manipulation of data during the mapping 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 transforming the source data into another format or pre/post-processing techniques are used. As part of this paper, we present a comparison framework for the state-of-the-art in converting CSV/Tabular data into RDF, where a key feature evaluated is transformation functions. We argue that existing approaches for transformation functions in such tools are complex – in number of steps and tools involved – and therefore not as traceable and transparent as one would like. We tackle these problems by defining a more generic, usable and amenable method to incorporate functions into uplift mapping languages, called FunUL. As proof of concept, we show an implementation of our method. Moreover, by using a real world Digital Humanities case study, we compare our approach with other approaches that we have identified to include transformation functions as part of the mapping for CSV/Tabular data.

Categories and Subject Descriptors

D.2.12 [**Interoperability**]: Data mapping

E.2 [**Data Storage Representations**]: Linked representations

General Terms

Algorithms, Design, Languages.

Keywords

Linked Data, Mapping, Data Manipulation.

1. INTRODUCTION

Significant amounts of data on the Web still resides in formats other than the Resource Description Framework[1] (RDF) 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 [11]. CSV/Tabular data (even though the delimiter is different, 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 this data format. In contrast, RDF provides one means to publish data and its meaning.

The process of converting data into RDF is called *uplift* [3]. As several 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 [9]. 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 [14]. 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 URIs). For some data formats this can be more-or-less straightforward. For example, with uplift tools for relational databases, such as R2RML [5] implementations, one can rely on SQL to provide the necessary transformation functions, whereas with RML [8], an R2RML extension with support for multiple data formats, XPath is used to transform XML data and JSONPath is used for JSON data. In many cases, however, relying on underlying technology[2] to undertake transformation might not be possible [6]. 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 the source data into another format or the use of pre/post-processing techniques. The

[1] `http://www.w3.org/TR/rdf11-concepts/`

[2] We define "underlying technology" in this paper as technology native to the source of the data.

use of these techniques, however, increases complexity – in relation to number of steps and tools involved. Furthermore, it 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 XML data type `xsd:gYear` (XML Schema specification[3]) 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 [4], proposed a method to include data transformations by the way of functions into Uplift Mapping Languages[4] in a generic, reusable and amenable way, here defined as FunUL. Our method integrates functions and mapping definitions. This allows data transformation and uplift of data into RDF to happen in a unified step. In addition, function definitions are reusable, being possible to call the same function multiple times with different parameters; and traceable. It is also possible to annotate them with provenance information, descriptions of the transformations defined and others. Moreover, functions can be applied to any data format.

In this paper we describe how we have implemented the proposed method by extending R2RML's vocabulary and RML's engine. We present an evaluation of our method by comparing, using a real world dataset, our approach with the only other approach that we have identified to include transformation functions in the mapping (that is KR2RML [16]).

The main contributions of this paper can be summarized as follows:

- a comparison framework for evaluating uplift tools in the process of converting CSV data into RDF;
- an evaluation of the state-of-the-art in CSV uplift tools using the proposed framework;
- a method to incorporate functions into uplift mapping languages and an implementation;
- evaluation of the method by comparing our approach to KR2RML.

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

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

Several solutions have been developed to uplift CSV data into RDF. In this section, we briefly introduce some of the tools in the state of the art that have support for CSV.

Uplift techniques can be described at a high level as to the type of approach that they support: mapping languages and additional software. **Mapping Languages (ML)** are declarative languages used to express customized mappings defining how non-RDF data should be represented in RDF. An 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 (AS)** 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.

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

Description	AS	ML
DataLift [15] is a tool 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[5] has an extension where CSV datasets can be imported into relational databases. After loading the data, mapping into RDF can be done using a wizard or R2RML mappings.	✓	✓
TopBraid Composer Maestro Edition[6] converts data into RDF with no customizations in a first step. A second step transforms the RDF data using SPARQL[7] queries and SPIN[8] rules.	✓	
DataOps [13] 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[9] has support for cleaning and transformation functions for many data formats. The uplift to RDF is available through RDF Refine[10], an extension to OpenRefine, allowing the mapping and interlinking of RDF datasets through a web interface.	✓	
KR2RML [16] 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, data cleaning and transformation functions.	✓	✓
RML [8] 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. RML also has an additional software tool to define RML map-	✓	✓

[3] `https://www.w3.org/TR/xmlschema-2/`

[4] We adapt the definition of Uplift Mapping Languages from [1] and define it as declarative languages for mapping non-RDF data sources into RDF vocabularies and OWL ontologies.

[5] `http://virtuoso.openlinksw.com/`

[6] `http://www.topquadrant.com/`

[7] `https://www.w3.org/TR/sparql11-query/`

[8] `http://spinrdf.org/`

[9] `http://openrefine.org/`

[10] `http://refine.deri.ie`

pings, RMLEditor [10].		
xR2RML [12], following RML, extends R2RML's vocabulary to support the uplift of various data formats into RDF.		✓
SML [17] currently supports relational databases and CSV datasets. SML is a mapping language based on SQL CREATE VIEWS and SPARQL construct queries.		✓
CSV2RDF[11] applies the use of a template file to determine how one row of the CSV dataset will be converted to RDF. The template file follows a Turtle like syntax.		✓
RDF-Tabular[12] 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 [13] as an abstract model. This model also allows the use of metadata through the Metadata Vocabulary for Tabular Data[14]. 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[15] is another fully compliant implementation of the CSVW specifications for the uplift of CSV data into RDF.		✓
Tarql[16] is a mapping language based on SPARQL CONSTRUCT queries to convert CSV datasets into RDF.		✓
Vertere-RDF[17] is a mapping language that uses a turtle syntax template file to define how CSV data should be represented in RDF.		✓

The next section presents a framework that allows for a more detailed comparison of the tools presented in this section.

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

In this section, we introduce a comparison framework to evaluate the state-of-the-art in CSV uplift, which was presented in Section 2. In [9], a comparison framework to evaluate relational databases to RDF tools on a feature-by-feature basis was proposed. We have adapted these features into the CSV uplift context, and propose 2 new features. One feature specific for CSV data: filtering; and a second important feature for the uplift process: reusability. The features 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 during the uplift process.
- **F3: Select.** A dataset may contain attributes that should not be a part of the RDF representation. Regarding CSV data, this feature allows the selection of columns 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 URI.** This feature allows the transformation of literals into valid URI'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 URI.
- **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 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 data types. 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.
- **F9: Named Graphs.** Named graphs are RDF datasets identified using an URI [11]. Applications to uplift data to RDF should support the conversion into a particular named graph.
- **F10: Blank Nodes.** Blank nodes are RDF statements with no RDF URI reference [11]. This feature allows the generation of blank nodes.
- **F11: Reusability (N).** This feature allows the serialization of the uplift process for further reuse.

To evaluate the tools presented in **Table 1**, we analyzed papers, documentation and tested a working implementation against the features defined in the comparison framework. In other words, we 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 the mapping of two related columns.

[11] https://github.com/clarkparsia/csv2rdf

[12] https://github.com/ruby-rdf/rdf-tabular

[13] https://www.w3.org/TR/tabular-data-model/

[14] https://www.w3.org/TR/tabular-metadata/

[15] https://github.com/theodi/csv2rdf

[16] https://github.com/tarql/tarql

[17] https://github.com/mmmmmrob/Vertere-RDF

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. Our comparison framework applied to CSV uplift tools can be seen in **Table 2**.

Table 2: Comparison summary for the state-of-the-art in CSV uplift.

	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

3.1 Discussion

In this section we discuss the state-of-the-art in CSV uplift using our comparison framework. The discussion is structured according to the features. In general, if an uplift tool is not mentioned, it means that the tool supports the feature.

- **F1 M:N Relationships.** DataOps does not have support for this feature. OpenRefine supports this feature by creating a new IRI with an existing IRI value, as it is not possible to select existing resources. DataLift has support for this feature but it can be very complex to redefine the RDF dataset after the direct mapping process. Top Braid Composer has support 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 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 tools analyzed have support for the selection of attributes that should be converted into RDF.
- **F4 Filter.** RML supports filters depending on the data format, so it is possible to use filters based on underlying technology. XPath is used for XML, for example. For CSV data, 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 first and then applies mappings to convert it into RDF, 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 data types, but it does not cover other filter options, such as comparing or combining different values from the dataset. Vertere-RDF also has partial support using regular expressions. Tarql supports this feature partially 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, this filter can be applied `FILTER (?value >= 10)`. Generally, most tools that support this feature rely on other technologies, such as SPARQL CONSTRUCT queries or regular expressions.
- **F5 Literal to URI.** This feature is supported by all applications, but it can be complex to define it using some tools. For example, in Datalift, you need to select an option for such transformation. Inside this option, it is necessary to define the dataset, options for reference 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 URI.
- **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, as custom functions are not supported without extending the code. 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 in the first step of the uplift process. Tarql supports some transformation functions through SPARQL queries. Most tools with additional software have support for transformation functions, such as OpenRefine and some mapping languages have pre-defined functions. 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, `xsd:dateTimeStamp` and others are not supported. All other tools have support for XML datatypes. Together with XML datatypes, some tools allow the definition of custom datatypes, such as OpenRefine.
- **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 allows one to publish data into a specific named graph or existing ones inside their tool. 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 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 different data structures.
- **F11 Reusability.** KR2RML allows the serialization of the process as an extended R2RML mapping. In this sense, the mapping contains one extra predicate with structured information as a string, making it difficult to create or modify the mapping without their editor. Furthermore, as shown in F7, transformation functions are not reusable. 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. KR2RML mapping language serializes the whole process applied to the input data as a string, what makes some parts of the mapping not reusable and the whole process dependent on their editor.

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 have to 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, data transformation functions are not reusable. It is not possible to use the same function, as functions in KR2RML do not have names or parameters. In this sense, a function update becomes difficult and prone to error – what we will show in Section 5.

The comparison framework allows the discussion of features needed for the uplift of data into RDF. In this paper we focus on tools with support for CSV files. A key feature analyzed by our comparison framework is transformation functions. As shown in our discussion, there is no underlying technology for CSV data and current approaches are not reusable or traceable. Current approaches rely on pre/post-processing techniques or on converting the data into another format when data transformation is needed. 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 by the use of our comparison framework is left as future work. In the next section, we present FunUL, a method to incorporate functions as part of uplift mapping languages.

4. FunUL - FUNCTIONS INTO UPLIFT MAPPING LANGUAGES

In this section, we describe FunUL, a method to incorporate data transformation by the way of functions into uplift mapping languages. Our method defines functions to be part of the mapping in a generic, reusable and amenable way. 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 URIs). In this paper, we elaborate upon previous work presented in [4].

The definition of functions as part of the mapping allows data transformation and uplift into RDF to happen in a unified step. Furthermore, functions are defined independently of the data mapping, being 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 others. 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 our method, *functions* have a *name* and a *body*. Each function declaration must have 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. A function body can perform data transformation and data validation tasks. The definition of parameters is optional. Every function defined in a function body must have a return statement. It is possible to return NULL or empty strings. An empty string should not generate any triples, as the semantic value of an empty string is NULL in CSV data format [18]. For example, as mentioned in Section 1, a function can be defined to transform years in BCE/CE notation from a dataset to use the XML data type `xsd:gYear`. We could define the name of this function as `yearTransformation`, the function body would have the implementation 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 passing fixed val-

ues 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 possible parameters (see Section 5.2).

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 Method Implementation

Our proof-of-concept extends R2RML's vocabulary and RML's engine by introducing construct for describing functions, function calls and parameter bindings. A specification[18] and an implementation[19] are available.

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 can be called using the property `rrf:functionCall`. This property refers to a `rrf:Function` using the property `rr:function`. Parameters are defined using `rrf:parameterBindings`. Examples of function definitions and function calls can be seen in Section 5.2.

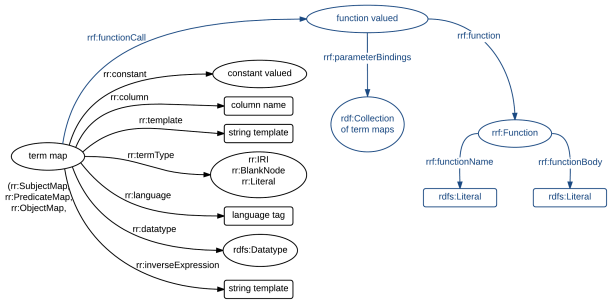

Figure 1. Properties of term maps based on the image from [5]

In RML – as an R2RML extension – 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. To implement FunUL, we introduce a *Function Valued* Term Map (see [5]). 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 data type. 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 a literal.

Analyzing the features from our comparison framework defined in Section 3, this implementation has support for the features F4 Filters and F7 Transformation Functions. F4 is supported by the use of functions. By extending RML, feature F9 Named Graphs, is still not supported by our 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.

The implementation loads functions using Java's Nashorn[20] JavaScript engine available in the javax.script package. We have chosen JavaScript for this implementation (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.

In our implementation, functions can be used for predicates (`rr:predicateMap`) or objects (`rr:objectMap`). By extending the implementation, functions could also be applied to subjects (`rr:subjectMap`). Because we represent functions using RDF, these can be shared. Furthermore, by extending RML's engine, functions defined using our implementation can be applied and reused for other input formats supported by RML.

5. EVALUATION

To evaluate the implementation, we compared our method with that of KR2RML using a real world Digital Humanities case study. This comparison evaluated how well both approaches could implement the required mappings.

5.1 The dataset

The dataset used in this evaluation comes from a project called Seshat: Global History Databank [19]. 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 [1]. 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

[18] `https://www.scss.tcd.ie/~crottija/funul/`

[19] `https://github.com/CNGL-repo/RMLProcessor`

[20] `https://blogs.oracle.com/nashorn/`

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 to RDF based on the new Seshat OWL Ontology.

5.2 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 has three parameters and it returns the value only if the first two parameters are equals. For example, a mapping to generate the predicate `seshat:ra` for the Seshat dataset, would call this function with parameters `rml:reference "Variable"`, `rr:constant "RA"` and `rml:reference "Value From"`. As mentioned before, the semantic value of an empty string is NULL. 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 "";
     }
""" ; .
```

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 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 "";
     }
""" ; .
```

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 `rml:reference`, the others are constants, `rr:constant`. In this sense, the function called is generic and could be reused passing other parameters.

```
<#DurationBeginning>
 rml:logicalSource [
    rml:source "data.csv";
    rml:referenceFormulation ql:CSV
 ];
 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 (
          [ rml:reference "Variable" ]
          [ rml:reference "Value From" ]
          [ rr:constant "Duration" ]
          [ rr:constant "0" ]
          [ rr:constant "-" ]
        ) ;
      ] ;
    ];
```

```
].
```

Listing 5: Calling the function `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: RDF output

5.3 KR2RML's approach

The same functions defined using our method could be created using KR2RML. 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 in a KR2RML mapping

5.4 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 is it 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. Moreover, we note that, for our use case and many others, transformations functions would be reused in the mapping.

Other problems with functions in KR2RML include, as it 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 their editor.

6. CONCLUSION

In this paper, we proposed a comparison framework to evaluate uplift tools applied to CSV datasets. Relying on one of the features – transformation functions – evaluated by our framework, we proposed a method to incorporate functions into uplift mapping languages. The general approach for data manipulation during the uplift process to convert 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 more transparent and traceable. We showed an implementation of our method by extending R2RML's vocabulary and RML's engine. Our evaluation applied the method to a real world use case and compared the use of functions as part of the mapping to KR2RML, the only other uplift tool identified to have this feature. Our evaluation showed that even though the whole process can be serialized using KR2RML, their functions are not reusable, making function updates complex and prone to error. Furthermore, KR2RML's mappings are stored as strings, what 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 of mappings difficult using other tools. In contrast, our method and implementation 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.

Future work includes extending the method to better describe functions; implementing the method by either extending another mapping language, as we did with RML, or implementing our own mapping language; investigating 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 [7]). Future work also includes the analysis of other features covered by our comparison framework and ways of dealing with such problems.

7. ACKNOWLEDGMENTS

This study is supported by: (i) CNPQ, National Counsel of Technological and Scientific Development – Brazil; (ii) the Science Foundation Ireland ADAPT Centre for Digital Content Technology (Grant 13/RC/2106); (iii) John Templeton Foundation grant to

the Evolution Institute [https://evolution-institute.org/project/seshat/]; (iv) the European Union Horizon 2020 ALIGNED [www.aligned-project.eu] (Grant 644055).

8. REFERENCES

[1] Bizer, C., Seaborne, A.: D2RQ - Treating Non-RDF databases as virtual RDF graphs. In: Proceedings of the 3rd international semantic web conference (ISWC2004). Volume 2004., Citeseer Hiroshima (2004).

[2] Brennan, R., Feeney, K., Mendel-Gleason, G., Bozic, B., Turchin, P., Whitehouse, H., Francois, P., Currie, T., Gohmann, S. Building the Seshat Ontology for a Global History Databank. In: The Semantic Web: ESWC (2016).

[3] Brennan, R., Feeney, K.C., Gavin, O.: Publishing Social Sciences Datasets as Linked Data: a Political Violence Case Study. In: Exploration, Navigation and Retrieval of Information in Cultural Heritage workshop (ENRICH 2013), Dublin, Ireland (2013).

[4] Crotti Junior, A., Debruyne, C., O'Sullivan, D.: Incorporating Functions in Mappings to Facilitate the Uplift of CSV Files into RDF. In: The Semantic Web: ESWC 2016 Satellite Events (2016).

[5] Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. (2012) https://www.w3.org/TR/r2rml/.

[6] Debruyne, C., O'Sullivan, D.: R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings. In: Workshop on Linked Data on the Web (2016).

[7] Dimou, A., De Nies, T., Verborgh, R., Mannens, E., and Van de Walle, R.: Automated Metadata Generation for Linked Data Generation and Publishing Workflows. In: Workshop on Linked Data on the Web (2016).

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

[9] Hert, M., Reif, G., Gall, H.C.: A comparison of RDB-to-RDF Mapping Languages. In: Proceedings of the 7th International Conference on Semantic Systems. I-Semantics '11, New York, NY, USA, ACM (2011) 25-32.

[10] Heyvaert, P., Dimou, A., Herregodts, A.L., Verborgh, R., Schuurman, D., Mannens, E., Van de Walle, R.: RMLEditor: A Graph-based Mapping Editor for Linked Data Mappings. In: The Semantic Web - Latest Advances and New Domains (ESWC 2016) (2016).

[11] Hitzler, P., Krotzsch, M., Rudolph, S.: Foundations of semantic web technologies. CRC Press, (2009).

[12] Michel, F., Djimenou, L., Faron-Zucker, C., Montagnat, J.: Translation of relational and non-relational databases into RDF with xR2RML. In: 11th Web Information Systems and Technologies (WEBIST) (2015).

[13] Pinkel, C., Schwarte, A., Trame, J., Nikolov, A., Bastinos, A.S., Zeuch, T.: Dataops: Seamless end-to-end anything-to-RDF data integration. In: The Semantic Web: ESWC 2015 Satellite Events (2015).

[14] Purohit, S., Smith, W., Chappell, A., West, P., Lee, B., Stephan, E., Fox, P.: Effective Tooling for Linked Data Publishing in Scientific Research. In: 2016 IEEE Tenth International Conference on Semantic Computing (ICSC) (2016)

[15] Scharffe, F., Atemezing, G., Troncy, F., Gandon, F., Villata, S., Bucher, B., Hamdi, F., Bihanic, L., Képéklian, G., Cotton, F., et al. Enabling linked data publication with the Datalift platform. In Proc. AAAI Workshop on Semantic Cities, 2012.

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

[17] Stadler, C., Unbehauen, J., Westphal, P., Sherif, M.A., Lehmann, J.: Simplified RDB2RDF Mapping. In: Workshop on Linked Data on the Web. (2015).

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

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