Towards a Uniform User Interface for Editing Mapping Definitions

Pieter Heyvaert, Anastasia Dimou
Ruben Verborgh, Erik Mannens, and Rik Van de Walle

Ghent University – iMinds – Multimedia Lab
pheyvaer.heyvaert@ugent.be

Abstract. Modeling domain knowledge as Linked Data is not straightforward for data publishers, because they are domain experts and not Semantic Web specialists. Most approaches that map data to its semantic representation still require users to have knowledge of the underlying implementations, as the mapping definitions remained, so far, tight to their execution. Defining mapping languages enables to decouple the mapping definitions from the implementation that executes them. However, user interfaces that enable domain experts to model knowledge and, thus, intuitively define such mapping definitions, based on available input sources, were not thoroughly investigated yet. This paper introduces a non-exhaustive list of desired features to be supported by such a mapping editor, independently of the underlying mapping language; and presents the RMLEditor as prototype interface that implements these features with RML as its underlying mapping language.

Keywords: Linked Data Mapping, Linked Data Mapping Interface, RML

1 Introduction

In recent years, the Web is evolving from the Web of Documents, to the Web of Data, where data can be interlinked (Linked Data [1]), according to the four Linked Data principles [2]. Since the Resource Description Framework (RDF) [3] allows adhering to these principles, different deployment approaches were introduced to map data to RDF. Most approaches rely on either *custom*, or *format-specific* implementations [4]. In both cases, the mapping definitions are tied to the corresponding implementation and, thus, new development cycles, executed by developers, are required to adjust them. Additionally, data publishers, who are domain experts, should be able to specify mapping definitions, and modify them at any time, because they possess the domain knowledge to be modeled.

Since data publishers are not developers or Semantic Web experts, the mapping definitions authoring should be decoupled from their execution. To this end, research has been conducted to define mapping languages, e.g., R2RML [5] and its extension RML [4, 6], which allow to separate the definitions from the execution, i.e., the implementation. R2RML is defined for mapping data in relational

databases, while RML generalizes its purpose to also support mappings for data in different formats.

Nevertheless, besides knowledge of the underlying mapping language, manually editing and curating definitions requires a substantial amount of human effort [7]. Therefore, the process of defining mappings should be facilitated. Research efforts to improve the usability of editing mapping definitions has led to two research topics: (i) (semi-)automatically generating mapping definitions [8] and (ii) abstracting the creation of mapping definitions from the definitions' syntax [8, 9, 10]. The former uses the source data for the (semi-)automatic generation of mapping definitions, by determining the data's semantics. These definitions can be edited afterwards by the user, if needed. The latter includes offering a graphical user interface (GUI), where the data, the mapping definitions and the resulting RDF dataset is integrated.

Most existing tools with a GUI follow a *step-by-step* workflow [9]. This way, adjustments to the previous steps are not straightforward and, in general, data publishers are distracted from the model overview, when they define a certain mapping definition. Others only provide an interface for explicitly editing the definitions, which requires knowledge of the underlying mapping language [9]. Moreover, there are solutions that do not allow to visualize the definitions without loading the input data [8]. This makes performing editing actions that do not require the input data impossible, e.g., updating a property that connects two resources. Last but not least, most solutions refer only to a *single data source* when modeling the RDF representation, while, the data required to complete the domain-model might be derived from multiple input sources.

In this paper we introduce a non-exhaustive list of desired features, independent of any underlying mapping language. They are addressed by a uniform mapping editor which aims to support data publishers to model domain-level knowledge. We implement such an editor, the RMLEditor, to offer these features to data publishers. Currently, the RMLEditor supports specifying mapping definitions for tabular-structured data, e.g., databases or data in CSV format. The underlying mapping language is the RDF mapping language (RML).

The remainder of the paper is structured as follows. We elaborate on related work in Section 2. Next, the desired features are listed in Section 3. In Section 4, we describe the RMLEditor and we end with conclusions and future work in Section 5.

2 Related Work

Editing mapping definitions independently of their execution requires abstracting them from their execution. Different mapping languages were defined in the past for this purpose. However, in most cases they are tightly coupled to a single format. For relational databases, besides the W3C-recommended R2RML [5], several other languages were defined [11]. Similarly, mapping languages were defined to support mapping data in CSV files and spreadsheets to the RDF data model. For instance, the declarative OWL-centric mapping language M^2 that maps data

from spreadsheets into the Web Ontology Language (OWL) [12], or the SPARQL-based Tarql[1] that maps data from CSV files to RDF. Apart from RML [4, 6], there are no uniform languages defined that cover different data structures and formats.

While GUIs are being built on top of mapping languages, such as the fluidOps editor [9] on top of R2RML, no thorough research has been conducted regarding the design of such GUIs. Sheet2RDF [10] is a platform for mapping spreadsheets to RDF. It allows data publishers to preview the source data, edit the mapping definitions and view the resulting triples. To describe the mapping rules, a PEARL [13] document is used. Editing the mapping definitions requires knowledge of the PEARL syntax. Hence, the data publishers need to understand the definitions' syntax. However, they are not Semantic Web experts, who have this understanding.

Earlier efforts to provide a GUI for mapping definitions editing lead to tools which allow users to include semantic annotations with the input data. An example is OpenRefine[2], which was originally focused on cleansing and transforming data. It allows to interlink data with other data by using web services and to upload the cleaned data to a central database. The RDF Refine[3] [14] extension is built to export data to the RDF data model. It offers a GUI for specifying the mapping definitions, using a (so-claimed) RDF graph. However, the graph is forced in a hierarchy-layout, which weakens the advantages of using a graph representation. In the same context, Karma[4] [8] follows a *semi-automatic* approach to map structured sources to ontologies in order to build semantic descriptions. Like OpenRefine, its main focus is on the input data. Karma uses Global-Local-As-View (GLAV) [15] rules to perform the mappings, which can be exported as R2RML or D2RQ mapping definitions. Due to the fact that the definitions can only be visualized with the input data, a data-independent overview of the mapping model is not possible.

Next to visualizing the editing of mapping definitions, a number of tools impose restrictions on the workflow followed by the user. Sengupta et al. [9] introduced the fluidOps editor for creating and editing R2RML mappings. It supports a *step-by-step* workflow that follows steps similar to creating a mapping document, while reducing the use of R2RML's vocabulary details. However, such a *step-by-step* workflow (i) restricts data publishers' editing options; (ii) makes altering parameters in previous steps difficult; and (iii) detaches editing mapping definitions from the model. Thus, data publishers might lose the global overview of the model, since related information is separated in different steps.

Pinkel et al. [7] adapted the original fluidOps editor to overcome flexibility limitations imposed by the *step-by-step* workflow to evaluate their different hypotheses regarding *ontology-driven* and *database-driven* approaches towards defining mappings. They concluded that an editor should support both

[1] *http://tarql.github.io/*
[2] *http://openrefine.org/*
[3] *http://refine.deri.ie/*
[4] *http://www.isi.edu/integration/karma/*

approaches. In another work of ours, we generalized these approaches to *schema-driven* and *data-driven*, and introduced the *model-driven* and *result-driven* approaches [16]. The *ontology-driven* approach can be applied to any schema, namely any combination of ontologies and/or vocabularies. With the *data-driven* approach, we consider any type of input and not only databases. In the case of the *model-driven* approach, firstly the domain can be modeled by generating abstract mappings, resulting in modeling the desired domain independently of the input data. With the *result-driven* approach, mappings can be generated based on the desired result of the mapping process.

Another tool that uses a *step-by-step* workflow to set up the mapping process is TopBraid Composer[5]. It is a Semantic Web modeling tool that facilitates the automated conversion of spreadsheets or Unified Modeling Language (UML) files into RDF. Similarly, a wizard application[6] to create mapping definitions for RML, uses a *step-by-step* workflow. However, in both cases, knowledge of the underlying mapping language is required, just like with the fluidOps editor. In both case, the workflow follows steps similar to directly editing a mapping document.

Last, Scharffe et al. [17] introduced the Datalift[7] platform to map raw data sources to semantically interlinked resources. The platform allows the addition of custom modules, to enable customer-specific features. However, Datalift also follows a *step-by-step* workflow which first applies a *direct-mapping* approach and then let data publishers redefine the semantic annotations. As the modifications are applied to the preliminary RDF model, generated after the direct mapping, the mapping definitions are interpreted as *SPARQL* `CONSTRUCT` queries.

3 Interface for Modeling Domain-level Knowledge

An RDF mapping editor aims to support data publishers to model domain-level knowledge as Linked Data using the, prevalent for this scope, RDF data model. To achieve this, an editor should support publishers in creating and editing mapping definitions to acquire the desired RDF representation, independently of the original state of the data. Below, we introduce a list of desired features to be supported by such a mapping editor.

Independence of mapping language Domain experts acting as data publishers are not Semantic Web experts. To model domain-level knowledge via mapping definitions, an editor should allow to edit them without confronting publishers with the syntax of the mapping editor's underlying language. For instance, a visualization for the definitions can be provided for modeling domain knowledge. The editor will interpret the visualization as statements in the language's syntax.

[5] *http://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/*
[6] *http://pebbie.org/mashup/rml*
[7] *http://datalift.org/*

Multiple data sources Data that is required to model domain-level knowledge might be derived from multiple data sources. Hence, mapping editors should support data publishers in defining mapping definitions referring to multiple data sources at the same time. For instance, two CSV files contain information about employees and projects. Connecting employees to their corresponding projects is only possible when definitions can be specified across the two files.

Heterogeneous data formats Mapping editors should not restrict data publishers from accurately modeling domain-level knowledge because of the original data format. They should rather enable editing mapping definitions independently of the source data format. For instance, information about the different teams in a company might be available via an XML file. Connecting the aforementioned employees to their corresponding team is only possible when definitions can be specified across the XML and CSV files.

Multiple ontologies and vocabularies Different ontologies and/or vocabularies, which model complementary or overlapping aspects of domain-level knowledge, are available. Hence, mapping editors should support data publishers in defining mapping definitions that annotate data with multiple ontologies and/or vocabularies at the same time. For instance, publishers should be able to provide information about people using the FOAF[8] ontology, and add their corresponding bibliographic information using the Bibliographic Ontology[9]. The desired result could be

```
<www.example.com/person/jd> a foaf:Person;
                            foaf:firstName "John";
                            foaf:lastName "Doe".
<www.example.com/journal/1> a bibo:Journal;
                            bibo:editor <www.example.com/person/jd>.
```

Multiple alternative modeling approaches Mapping editors should enable and support multiple alternative modeling approaches [7] and allow data publishers to choose the most adequate one for their needs. To be more precise, as defined by the data-driven approach, publishers might start with the input data that is desired to be semantically annotated. For instance, with tabular data, the columns to be mapped can be selected, followed by adding the correct classes and properties from the used ontologies and/or vocabularies. However, not all required columns need to be selected before the semantics can be added. Alternatively, given a model, described by an ontology, they might want to associate the input data with the different elements of the model, as defined by the schema-driven approach. For instance, after modeling the domain knowledge using an ontology, the user selects which columns, from the tabular data, represent which parts of the knowledge that is being modeled.

[8] *http://www.foaf-project.org/*
[9] *http://bibliontology.com/*

Non-linear workflows Modeling domain-level knowledge involves multiple factors – data, schema(s) and mapping definitions – that have an influence on each other. A linear workflow separates these factors and obscures their relationships, by using ordered steps that need to be followed by the data publishers. Non-linear editing allows publishers to keep an overview of the mapping model and its relationships. For instance, when adding the semantic annotations to the model, publishers might find that the data of certain column of a CSV file is missing in the model. The non-linear workflow allows to integrate that data into the model without the need to redo (parts of) the mapping process.

Independence of execution Editing mapping definitions lies beyond the scope of their execution. Thus, mapping editors should be able to export the set of mapping definitions, specified by the data publishers, through the user interface. As a result, further processing or execution, outside the mapping editor, improves the definitions' interoperability and reusability. For instance, when the definitions are created locally using an editor, they can be executed on a server without further need of the editor.

4 RMLEditor

In this section, we introduce our RMLEditor. First, we discuss its underlying language RML. It is used because it is a mapping language that can express uniform mapping definitions on top of heterogeneous data. Thus, RML can express what data publishers define via the editor interface (see Section 4.1). Then, we elaborate on the Graphical User Interface of the RMLEditor (in Section 4.2), based on the desired features. Next, we describe how data publishers interact with the RMLEditor (in Section 4.3), followed by how the mapping definitions are interpreted as RML statements (in Section 4.4). Last, we elaborate on how each desired feature is supported by the RMLEditor (in Section 4.5). A screencast demonstrating the RMLEditor can be found at *http://rml.io/RMLeditor*.

4.1 RDF Mapping Language (RML)

RML [4, 6] is the RMLEditor's underlying language. It is a generalization of R2RML [5] to support mapping definitions referring to data in different formats. RML is defined as a superset of R2RML, with the goal to extend its applicability and broaden its scope. While R2RML is limited to homogeneous data sources (i.e., databases), RML supports multiple heterogeneous sources. Although it adopts the same syntax, it can be enhanced with case-specific extensions. Next, we will elaborate on an example. First, the data is available in the CSV file 'person.csv'. This results in the following `Logical Source`

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

<#PersonMapping>
  rml:logicalSource [
    rml:source "/path/to/person.csv";
    rml:referenceFormulation ql:CSV ].
```

The location of the file is specified by `rml:source`. Because it is a CSV file, we use the correct reference formulation using `rml:referenceFormulation`. Next, the subject of the RDF triples is determined by the `Subject Map`, where `rr:template` explains how the IRI is constructed. In the example the value of the 'Identification' column is used. The class of subject is determined by `rr:class`. In the example, the class is set to `foaf:Person`.

```
<#PersonMapping>
  rr:subjectMap [
    rr:template "http://www.example.com/person/{Identification}";
    rr:class foaf:Person ].
```

Using the column 'Name' (via `rrmlreference` in an `Object Map`), the family name of a person (`foaf:familyName`) can be added via `rr:predicate` to a `Predicate Object Map`. This results in

```
<#PersonMapping>
  rr:predicateObjectMap [
    rr:predicate foaf:familyName;
    rr:objectMap [
      rml:reference "Name" ]].
```

The RMLProcessor[10] can be used to apply the mapping definitions on the source data. It is integrated in the RMLEditor to support the execution of definitions.

RML was chosen because it can implement all desired features that are related to the underlying mapping language. Currently, it is the only language that offers uniform mapping definitions and natively supports multiple and/or heterogeneous data sources. Using the interface with a mapping language that does not support multiple and/or heterogeneous data sources would restrict data publishers from using all their available data.

4.2 RMLEditor's Graphical User Interface

There are four aspects that contribute when modeling knowledge: (i) the input data, (ii) the schemas, (iii) the mapping definitions and (iv) the resulting semantic representation. This is translated in the GUI of the RMLEditor to three different panels: (i) the *Input Panel*, (ii) the *Modeling Panel* and (iii) the *Results Panel*. The first panel shows a sample of each input data source, the second panel allows data publishers to edit the mapping definitions, including the used schemas, and the third panel displays the resulting dataset. These three panels are aligned next to each other as shown in Figure 1. Additionally, (re)loading the sources and editing the definitions can occur independently and interchangeably.

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

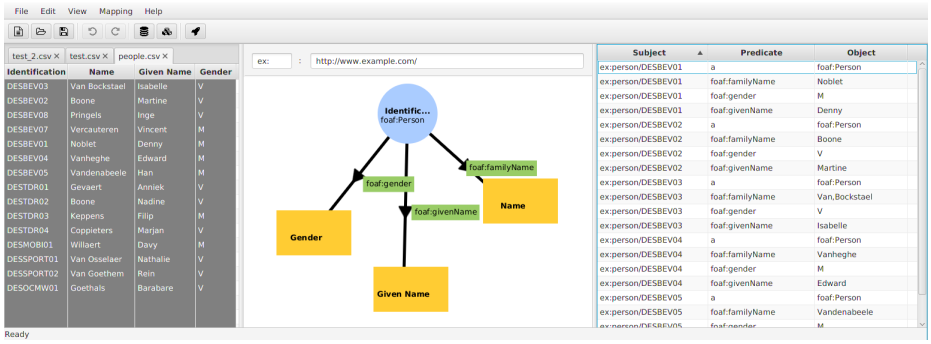

Fig. 1. The *Input Panel*, *Modeling Panel* and *Results Panel* are aligned next to each other in the RMLEditor.

When creating a GUI to provide a visualization for ontologies, a common approach is the use of graphs [18]. The Visual Notation for OWL Ontologies [19] (VOWL) defines a visual language, using graphs, for the representation of ontologies. Nodes are used to represent the classes and the datatypes, while edges represent the relationships between the nodes. For the visualization of the mapping definitions in the *Modeling Panel*, we use graphs to represent how the RDF dataset will be generated. Similar to ontology visualizations, the nodes represent the mapping definitions that generate the RDF terms (URIs, Blank Nodes and Literals), while the edges represent the relationships between them, namely the properties that associate them.

In the *Modeling Panel*, the references to the data (i.e., the column names in the case of databases, or header names in the case of CSV files), are entwined together with information about their class or datatype, represented by the nodes. The edges represent the properties that connect two types of data. By doing so, we adhere as much as possible to the VOWL specification. For instance, as shown in Figure 2, people (visualized as a node; represented by the column 'Identification'; Figure 2a) of the class `foaf:Person` are connected to their family name (visualized as a node; represented by the column 'Name'; Figure 2b) by the property `foaf:familyName` (visualized as an edge; Figure 2c). Similarly, people are connected to their given name (using the property `foaf:givenName`; represented by the column 'Given Name') and gender (using the property `foaf:gender`; represented by the column 'Gender').

4.3 Interacting with the RMLEditor

Having different panels in the RMLEditor allows data publishers to follow different approaches when generating mapping definitions. For instance, data publishers can first use the *Input Panel* to begin the modeling. *Nodes* (Figures 2a and 2b) can be added by right-clicking on a column and selecting *New Resource* for adding a resource or a blank node, or *New Literal* for adding a literal, with an optional datatype and language annotation. The result of this action is a new

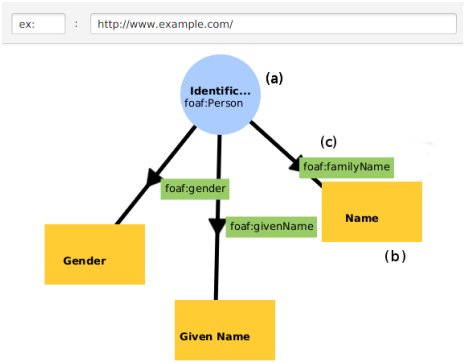

Fig. 2. Modeling panel of the RMLEditor.

```
<#PersonMapping>
(d) rml:logicalSource [
        rml:source "/path/to/person.csv";
        rml:referenceFormulation ql:CSV
    ];
(a) rr:subjectMap [
        rr:template "http://www.example.com/
                     person/{Identification}";
        rr:class foaf:Person
    ];
(c) rr:predicateObjectMap [
        rr:predicate foaf:familyName;
(b)     rr:objectMap [
            rml:reference "Name"
        ]
    ].
```

List. 1. RML statements

node in the *Modeling Panel* on which further mapping actions can be performed, e.g., add its class if it is a resource or its datatype if it is a literal. The *Node* contains a reference to the corresponding data fraction. As data publishers are not restricted to a single source, nodes might be generated from any of the input sources.

Alternatively, data publishers can use the *Modeling Panel* directly to edit the mapping definitions. Right-clicking on an empty location on that panel allows to create a new *Resource Node* (Figure 2a), *Literal Node* (Figure 2b) or *Edge* (Figure 2c), by clicking on *New Resource*, *New Literal* or *New Edge*, respectively. As soon as a *Node* is created, data publishers can associate it with the source data and further edit its semantic annotations, namely, edit its class, datatype, language and base IRI. If an edge is created, publishers can set the predicate used to define the relationship between two nodes. As most entities in the resulting dataset have the same base IRI, the RMLEditor allows publishers to define the base IRI and, optionally, the corresponding prefix, in the *Modeling Panel*.

The RMLEditor allows data publishers to execute the mapping on either the sample or the complete input data. For the sample input, this is done by clicking on the *Run Mapping* button in the toolbar. The resulting RDF dataset is visible in the *Results Panel*. A part of the mapping definitions is sufficient to have a first idea of the resulting dataset. This allows to easily detect errors early in the mapping process. Thanks to the fact that the panels are aligned next to each other, errors can be directly addressed using the *Model Panel*. When data publishers want to use their complete screen estate to work on the modeling, they can easily hide the other panels thanks to the *Hide Input Panel* and *Hide Results Panel* buttons in the toolbar. When the mapping definitions are complete, data publishers are able to export the mapping definitions.

The fact that the RMLEditor provides the possibility to simultaneously present all panels, allows to visualize the relationships between the different elements of the panels. To be more precise, when a node on the *Modeling Panel* is associated to a data fraction, that fraction is highlighted in the *Input Panel*.

Table 1. The table lists the features and shows which features each panel supports.

Features	***Panel***		
	Input	***Modeling***	***Results***
Independence of language		x	
Multiple data sources	x		
Heterogeneous data formats	x		
Multiple ontologies and vocabularies		x	
Multiple alternative modeling approaches	x	x	x
Non-linear workflows	x	x	x
Independence of execution		x	

4.4 Interpreting Graphical Mapping Definitions as RML Statements

The mapping definitions modeled in the *Modeling Panel* are translated into RML statements, which can then be further processed or executed using the RMLProcessor. Below, we explain how the definitions in Figure 2 are interpreted as the RML statements in Listing 1 (a detailed explanation of the statements can be found in Section 4.1).

First, the *Resource Node* (Figure 2a) is interpreted as a Subject Map (Listing 1a) and based on this, a Triples Map (<#PersonMapping> in Listing 1) is formed. The Subject Map will generate the subject of the triple. The class foaf:Person is interpreted as a class statement, using rr:class. The IRI of the resource is created based on the value of the 'Identification' column of each person, as defined by rr:template of the Subject Map.

Next, the *Literal Node* representing the (family) name of the person (Figure 2b) is translated to an Object Map (Listing 1b), which specifies how the object of a triple is generated. Subsequently, the *Edge* that states that the person and its name are connected with foaf:familyName (Figure 2c) is translated to a Predicate Map. A Predicate Object Map (Listing 1c) associates each Object Map to its Predicate Map. When a *Resource Node* is connected to another *Resource Node*, a Referencing Object Map is formed for the current Triples Map and another Triples Map is formed having the latter *Resource Node* as its Subject Map. An independent *Literal Node* is not possible.

Finally, as seen in Listing 1d, a Logical Source defines the source of the data (via rml:source) and how the source is referenced (via rml:referenceFormulation) based on the data format, which in the example's case is CSV. This information is derived from the *Input Panel*.

4.5 Implementing the Desired Features

The GUI of the RMLEditor implements the desired features of Section 3. Below, we will elaborate on each feature how it is facilitated. A summery of which features are supported by which panels can be found in Table 1.

Independence of mapping language The challenge is to abstract the mapping definitions from the syntax, namely the underlying mapping language. The *Modeling Panel* achieves this by visualizing the definitions using graphs (Section 4.2). When the definitions need to be executed on the input data, the graphs are interpreted as RML statements (Section 4.4). Additionally, when implementing the RMLEditor, we used modular programming. Thus, a new module can be added to use another mapping language to interpret the model.

Multiple data sources Multiple data sources are visualized in the *Input Panel*, using a tab to represent each data source. A tab will contain a sample of the data. When data publishers add an new source, a tab will be added to the panel.

Heterogeneous data formats Each tab of the *Input Panel* will visualize the data source in a way that is best suited for the format of the data. Currently, only tabular data, e.g., data in databases or in CSV format is supported by the RMLEditor. However, new modules can be added to offer new or other visualizations for different formats.

Multiple ontologies and vocabularies Using the *Modeling Panel*, publishers can set the class of a resource node or the datatype of a literal node. They are not restricted to which ontologies or vocabularies can be used, and they can be changed at any time during the mapping process. The RMLEditor aids publishers in finding appropriate classes, datatypes and properties. In order to achieve that we integrated the use of Linked Open Vocabularies (LOV)[11]. Prefix.cc[12] is used to help publishers in looking up namespaces of the used schema(s).

Multiple alternative modeling approaches By using three panels, data publishers can decide, based on their current need, which approach they prefer to follow. By first creating nodes based on the data in the *Input Panel*, publishers are able to apply the data-driven approach. By first defining a model, described by an ontology, in the *Modeling Panel*, publishers can subsequently associate the input data with the different elements of the model, as described by the *schema-driven* approach.

Non-linear workflows By showing the three panels simultaneously, data publishers have an overview of the mapping process. Thus, editing different elements of the process can be done in the corresponding panel without the need for tracing back previously completed steps.

[11] *http://lov.okfn.org/dataset/lov*
[12] *http://prefix.cc/*

Independence of execution At any point during the mapping progress data publishers are able to export the mapping definitions as RML statements, as facilitated by the *Modeling Panel*. Subsequently, the definitions can be edited and/or executed without the need of the RMLEditor.

5 Conclusion and Future Work

In this paper, we present a set of desired features to be considered during design and implementation of a uniform mapping editor. By doing so, data publishers, who are domain experts, are able to model knowledge derived from heterogeneous data. The proposed editor, the RMLEditor, offers these features, using RML as its underlying mapping language. The RMLEditor's interface is implemented based on separate panels, which are aligned next to each other.

Modeling domain-level knowledge with Linked Data is a task that should be performed by data publishers, who are domain experts, rather than Semantic Web specialists. There are four aspects that contribute to this modeling: (i) the input data, (ii) the schemas, (iii) the mapping definitions and (iv) the resulting semantic representation. Mapping editors and their GUIs should enable data publishers to build their model on top of any of them, while incorporating non-linear workflows and multiple approaches. These workflows and approaches give publishers the freedom to use the editor as it best fits the problem at hand. In order to achieve that, designing an interface with multiple panels, as the one proposed and implemented in the RMLEditor, seems to be an adequate solution.

Moreover, the modeling should be detached from the syntax of the underlying mapping language and/or the format and number of input sources. In the RMLEditor, the *Modeling Panel*, which uses a graph visualization for the mapping definitions, allows this detachment.

In the future, we will add support for non-tabular-structured data to the RMLEditor. We will conduct an analysis to determine which mapping definitions generation approaches need to be supported by the RMLEditor. Last, evaluation based on user study, is planned to validate the RMLEditor.

Acknowledgements The described research activities were funded by Ghent University, iMinds, the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research Flanders (FWO Flanders), and the European Union.

References

[1] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009.

[2] Tim Berners-Lee. Linked data, 2006, 2006. URL *http://www.w3.org/DesignIssues/LinkedData.html*.

[3] Dan Brickley and R. V. Guha. RDF Schema 1.1. Working group recommendation, W3C, February 2014. URL *http://www.w3.org/TR/rdf-schema/*.

[4] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Workshop on Linked Data on the Web*, 2014.

[5] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. Working group recommendation, W3C, September 2012. URL *http://www.w3.org/TR/r2rml/*.

[6] Anastasia Dimou, Miel Vander Sande, Jason Slepicka, Pedro Szekely, Erik Mannens, Craig Knoblock, and Rik Van de Walle. Mapping Hierarchical Sources into RDF using the RML Mapping Language. In *Proceedings of the 8th IEEE International Conference on Semantic Computing*, 2014.

[7] Christoph Pinkel, Carsten Binnig, Peter Haase, Clemens Martin, Kunal Sengupta, and Johannes Trame. How to best find a partner? An evaluation of editing approaches to construct R2RML mappings. In *The Semantic Web: Trends and Challenges*, pages 675–690. Springer, 2014.

[8] Craig A Knoblock, Pedro Szekely, José Luis Ambite, Aman Goel, Shubham Gupta, Kristina Lerman, Maria Muslea, Mohsen Taheriyan, and Parag Mallick. Semi-automatically mapping structured sources into the semantic web. In *The Semantic Web: Research and Applications*, pages 375–390. Springer, 2012.

[9] Kunal Sengupta, Peter Haase, Michael Schmidt, and Pascal Hitzler. Editing R2RML mappings made easy. In *Proceedings of the 12th International Semantic Web Conference: Posters and Demos*, pages 101–104. 2013.

[10] Manuel Fiorelli, Tiziano Lorenzetti, Maria Teresa Pazienza, Armando Stellato, and Andrea Turbati. Sheet2rdf: a Flexible and Dynamic Spreadsheet Import&Lifting Framework for RDF. In *Current Approaches in Applied Artificial Intelligence*, pages 131–140. Springer, 2015.

[11] Matthias Hert, Gerald Reif, and Harald C. Gall. A comparison of RDB-to-RDF mapping languages. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, pages 25–32. ACM, 2011.

[12] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). Working group recommendation, W3C, December 2012. URL *http://www.w3.org/TR/owl-syntax/*.

[13] Maria Teresa Pazienza, Armando Stellato, and Andrea Turbati. Pearl: Projection of annotations rule language, a language for projecting (uima) annotations over rdf knowledge bases. In *LREC*, pages 3828–3835, 2012.

[14] Fadi Maali, Richard Cyganiak, and Vassilios Peristeras. Re-using cool uris: Entity reconciliation against lod hubs. *LDOW*, 813, 2011.

[15] Marc Friedman, Alon Y Levy, Todd D Millstein, et al. Navigational plans for data integration. *AAAI/IAAI*, pages 67–73, 1999.

[16] Pieter Heyvaert, Anastasia Dimou, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. Approaches for generating mappings to RDF. In *Proceedings of the 14th International Semantic Web Conference: Posters and Demos*, October 2015.

[17] François Scharffe, Ghislain Atemezing, Raphaël Troncy, Fabien Gandon, Serena Villata, Bénédicte Bucher, Fayçal Hamdi, Laurent Bihanic, Gabriel Képéklian, Franck Cotton, et al. Enabling linked data publication with the Datalift platform. In *Proc. AAAI workshop on semantic cities*, 2012.

[18] Akrivi Katifori, Constantin Halatsis, George Lepouras, Costas Vassilakis, and Eugenia Giannopoulou. Ontology visualization methods - a survey. *ACM Computing Surveys (CSUR)*, 39(4):10, 2007.

[19] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. VOWL 2: User-oriented visualization of ontologies. In *Proceedings of the 19th International Conference on Knowledge Engineering and Knowledge Management (EKAW '14)*, volume 8876 of *LNAI*, pages 266–281. Springer, 2014.