Towards Semi-automatic Generation of R2R Mappings

Valéria M. Pequeno[1,2], Vânia M. P. Vidal[3] and Tiago Vinuto[3]
[1]*TechLab, Departamento de Ciências e Tecnologias,*
Universidade Autónoma de Lisboa Luís de Camões, 1150-023 Lisboa, Portugal
[2]*INESC-ID, Lisboa, Portugal*
[3]*Departamento de Computação, Universidade Federal do Ceará, Fortaleza, Brazil*

Keywords: Mapping Patterns, RDF-to-RDF Mapping, R2R Mapping, Mapping Assertion, RDF Model, Ontologies.

Abstract: Translating data from linked data sources to the vocabulary that is expected by a linked data application requires a large number of mappings and can require a lot of structural transformations as well as complex property value transformations. The R2R mapping language is a language based on SPARQL for publishing expressive mappings on the web. However, the specification of R2R mappings is not an easy task. This paper therefore proposes the use of mapping patterns to semi-automatically generate R2R mappings between RDF vocabularies. In this paper, we first specify a mapping language with a high level of abstraction to transform data from a source ontology to a target ontology vocabulary. Second, we introduce the proposed mapping patterns. Finally, we present a method to semi-automatically generate R2R mappings using the mapping patterns.

1 INTRODUCTION

Nowadays, there is a large number of datasets (of the different domain) published on the Web. These datasets are linked and published in RDF formats and usually are available in the Linked Open Data (LOD), creating a global data space known as Web of Data. The principles of the Web of Data emphasize the definition of the conceptual structure of the data through the re-use of known ontologies. Thus, the need for alignment between conceptual schemas is minimized. However, Linked Data sources normally use different vocabularies to represent data about a specific type of object. For instance, **DBpedia**[1] and **Music ontology**[2] use their own proprietary vocabularies to represent data about musical artists. The resulting data heterogeneity is a major obstacle to build useful Linked Data applications. Thus, the most of data that is available on the web need to be integrated and exchanged in a proper way.

Translating data from Linked Data sources (*the source ontology*) to the vocabulary that is expected by a linked data application (*the target ontology*) requires a large number of mappings. There are some approaches in the literature that focus on the development of formalisms to represent correspondences between ontological entities such as classes and properties (see (Shvaiko and Euzenat, 2013) for a survey). However, for the development of a linked data application, it is not enough to say, for example, that one class corresponds to another. We should focus on capturing information about which entity can be transformed into another and how this can be done. This type of mapping between the ontologies usually requires lots of structural transformations as well as complex property value transformations using, possibly, various functions.

The *LDIF framework* (Schultz et al., 2011) proposes the R2R mapping language for specifying mappings between RDF schemas. R2R is a very expressive language with a SPARQL-based syntax. However, the R2R framework[3] only provides general error messages that do not help the user in the identification of syntax errors in the mappings. This occurs because LDIF does not address the problem of how the mappings are defined, providing only the language to define the mappings. It calls for the development of methods and tools to support the deployment of mappings using R2R.

As the main contribution of this paper, we suggest a semi-automated pattern-based approach to ge-

[1]http://dbpedia.org/resource/

[2]http://musicontology.com/

[3]http://r2r.wbsg.de/

Pequeno, V., Vidal, V. and Vinuto, T.
Towards Semi-automatic Generation of R2R Mappings.
DOI: 10.5220/0006801602370244
In *Proceedings of the 20th International Conference on Enterprise Information Systems (ICEIS 2018)*, pages 237-244
ISBN: 978-989-758-298-1
Copyright © 2019 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

nerate R2R mappings. Even though different researchers were concerned with similar topics (see (Ritze et al., 2009; Scharffe et al., 2014)), to our knowledge none of the existing works present the constraints between different mappings to guarantee that the whole set of mappings between the target and the source ontologies generates correct instances. In addition, our work is the first to propose the semi-automatic generation of R2R mappings. Another contribution of this paper is the definition of Mapping Assertions (MAs) (informally addressed in (Pequeno et al., 2015)) as a convenient way to manually specify mappings between RDF vocabularies.

The rest of the paper is organized as follows. Section 2 briefly presents the R2R language and a motivating example. Section 3 introduces our formalism to define mappings. Section 4 briefly presents the proposed mapping patterns. Section 5 points out how semi-automatically to generate R2R mappings by applying mapping patterns. Section 6 discusses the related work. Finally, Section 7 presents our conclusions and future works.

2 MOTIVATING EXAMPLE

R2R is a declarative language based on SPARQL for publishing mappings between different RDF vocabularies. A R2R mapping refers to a class mapping (the **r2r:classMapping**) or a property mapping (the **r2r:propertyMapping**) to retrieve data from the source ontology and translate it to a target ontology vocabulary.

Every R2R mapping has both clauses: **r2r:sourcePattern** and **r2r:targetPattern** (like in a SPARQL CONSTRUCT clause). The source pattern is matched against Web Data and binds values to a set of variables. It may include almost all expressions that are possible in a SPARQL WHERE clause. Slightly talking, R2R source patterns correspond to the source pattern of our Mapping Rule (MR). The target pattern is used to produce triples in the target vocabulary. It corresponds to the target pattern of our MR. An R2R mapping may consist of multiple target patterns but has a single source pattern. It is easier to understand these concepts using an example.

Let us consider a linked data application about music that describe properties and concepts using the **MyMusic** ontology (the *target ontology*) and integrate data from both sources **DBpedia**[4] and **MySpace**[5]. The fragment of the **DBpedia** (shown in Fig. 1) depicts information about artists and related aspects. **MySpace** provides part of a RDF representation of MySpace users. A fragment of **MySpace** is shown in Fig. 2. **MyMusic** ontology provides information about musical artists and reuses terms from well-known vocabularies, such as: FOAF (**Friend of a friend**)[6] and MO (**Music ontology**). We use the prefix "ex" for new terms defined in the **MyMusic** ontology. For example, *ex:labelName* keeps the name of the record label. A fragment of **MyMusic** is shown in Fig. 3.

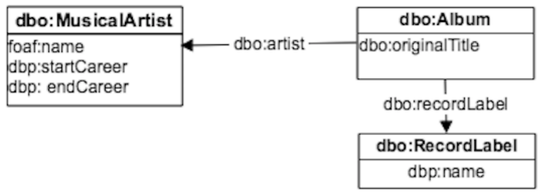

Figure 1: A simplified fragment of the **Dbpedia** ontology.

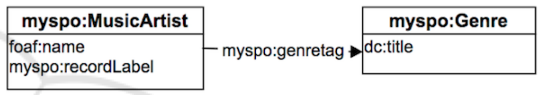

Figure 2: A simplified fragment of the **MySpace** ontology.

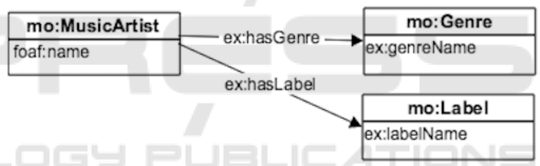

Figure 3: A simplified fragment of the **MyMusic** ontology.

In order to populate **MyMusic** ontology, we need specify mappings between **MyMusic** ontology and both **Dbpedia** and **MySpace** ontologies. These mappings should indicate how we can transform triples of the source ontologies in triples of the target ontology. For example, Fig. 4 shows an example of mapping between *mo:Genre* and *myspo:Genre* using the R2R mapping language. This is a very simple mapping in which all instances of *myspo:Genre* (i.e. triples of the form (*s rdf:type myspo:Genre*)) will be transformed in triples of the form (*s rdf:type mo:Genre*) (Lines 04 and 05 in Fig. 4). The clause **r2r:prefixDefinitions** is used to abbreviate URIs inside the **r2r:sourcePattern** or **r2r:targetPattern**. The instance variable ?SUBJ is used in every source pattern and is reserved for representing the instances that are the focus of the mapping.

R2R mappings also can have a clause **r2r:transformation** that defines how the values in **r2r:targetPattern** are transformed and a clause

[4]http://dbpedia.org/ontology/ and http://dbpedia.org/property/.

[5]http://purl.org/ontology/myspace/

[6]http://xmlns.com/foaf/0.1/

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

```
01. mp:mySpace_Genre_to_myMusic_Genre
02. a r2r:ClassMapping;
03.   r2r:prefixDefinitions "mo:<···>.myspo:<···>";
04.   r2r:sourcePattern "?SUBJ a myspo:Genre;
05.   r2r:targetPattern "?SUBJ a mo:Genre".
```

Figure 4: R2R mapping between *mo:Genre* and *myspo:Genre*.

r2r:mappingRef that refers to a **r2r:classMapping** defined before. For example, Fig. 5 shows the mapping between *mo:Label* and *myspo:recordLabel*, a more complex mapping than that showed in Fig. 4 because a class in an ontology corresponds to a property in another. When mapping *mo:Label* with *myspo:recordLabel*, a new URI must be generated based on both the URI and the property value of *myspo:recordLabel*. Thus, the subject triple of the *mo:Label* is the new URI generated in Line 06. For simplicity, here the new URI *u* is obtained by using the functions concat() and xpath:encode-for-uri().

```
01. mp:Label_to_recordLabel
02. a r2r:ClassMapping;
03.   r2r:prefixDefinitions "mo:<···>.myspo:<···>";
04.   r2r:sourcePattern "?SUBJ a myspo:MusicArtist;
                         myspo:recordLabel ?r;
05.   r2r:targetPattern "?u a mo:Label";
06.   r2r:transformation "?u= concat(?SUBJ,
                          xpath-encode-for-uri(?r))" .
```

Figure 5: R2R mapping between *mo:Label* and *myspo:recordLabel*.

Fig. 6 shows the mapping between *ex:labelName* and *myspo:recordLabel*, a mapping between datatype properties. The **r2r:mappingRef** (Line 04) makes references to the class mapping mp:Label_to_recordLabel shown in Fig. 5. It is used mainly to reduce redundancy: the source pattern of mp:Label_to_recordLabel is joined with the source pattern of mp:labelName_to_recordLabel. Lines 05 and 06 specify how *myspo:recordLabel* resources of *myspo:MusicArtist* are transformed to the format of the target triple (i.e., the triple of *ex:labelName*). Also, in this mapping, we need to generate a URI (Line 07), which must be the same generated in the mapping shown in Fig. 5.

R2R is ready for use. This means that we can generate RDF triples to a target ontology from R2R mappings and publish these mappings on the web. Besides, because R2R is based on SPARQL, it is easy to the users understand the generated mapping, since more and more developers know SPARQL. However, complex mappings, such as those presented in Figures 5 and 6, require expertise on the involved ontologies, as well as on the R2R language used to define the mappings. In addition, the manual definition of mappings can be tedious and error-prone. A library of mapping patterns will facilitate the generation of R2R mappings by providing templates modelling complex mappings such as the ones given above.

```
01. mp:labelName_to_recordLabel
02. a r2r:PropertyMapping;
03.   r2r:prefixDefinitions "ex:<···>.myspo:<···>" ;
04.   r2r:mappingRef mp:Label_to_recordLabel ;
05.   r2r:sourcePattern "?SUBJ myspo:recordLabel ?r" ;
06.   r2r:targetPattern "?u ex:labelName ?r" ;
07.   r2r:transformation "?u= concat(?SUBJ,
                          xpath-encode-for-uri(?r))" .
```

Figure 6: R2R mapping between *ex:labelName* and *myspo:recordLabel*.

3 MAPPING RDF TO RDF

3.1 Mapping Rules

In this section, we briefly present a mapping formalism, based on rules, to transform instance data from a source ontology to the target ontology vocabulary.

Our formalism is much simpler than familiar rule-based languages, such as SWRL[7], or mapping languages, such as R2R (Bizer and Schultz, 2010), but it suffices to capture expressive mappings. Also, the formalism incorporates *concrete domains* (Lutz, 2002) to capture concrete functions, such as “string concatenation”, required for complex mappings, and concrete predicates, such as “less than”, to specify restrictions.

A *vocabulary* $\mathbf{V}$ is a set of *classes* and *properties*. An *ontology* is a pair $\mathbf{O}$=($\mathbf{V}$,Σ) such that $\mathbf{V}$ is a vocabulary and Σ is a finite set of formulae in $\mathbf{V}$, the *constraints* of $\mathbf{O}$.

Let $\mathbf{V_T}$ be a *target vocabulary* and $\mathbf{O_S}$=($\mathbf{V_S}$,$\Sigma_\mathbf{S}$) be a *source ontology* with $\mathbf{V_S}$ and $\Sigma_\mathbf{S}$ being, respectively, the source vocabulary and the set of constraints of $\mathbf{O_S}$. Let $\mathcal{X}$ be a set of *variables*. Let $\mathcal{C}$ be a first-order alphabet consisting of a set $\mathcal{F}$ of function symbols and a set $\mathcal{P}$ of predicate symbols, respectively called *concrete function symbols* and *concrete predicate symbols*. The 0-ary function symbols are called *constants*, which include IRIs[8] and datatype values. We assume that the symbols in $\mathcal{C}$ have a fixed interpretation. Lastly, we assume that $\mathcal{X}$ and $\mathcal{C}$ are mutually disjoint and that $\mathcal{C}$ is disjoint from $\mathbf{V_T}$ and $\mathbf{V_S}$.

[7] https://www.w3.org/Submission/SWRL/

[8] Internationalized Resource Identifier.

A *term* is an expression recursively constructed from function symbols, constants, and variables, as usual. A *literal* is an expression of one of the forms:

- a *class literal* of the form $C(t)$, where C is a class in $\mathbf{V_T} \cup \mathbf{V_S}$ and t is a term;
- a *property literal* $P(t,u)$, where P is a property in $\mathbf{V_T} \cup \mathbf{V_S}$ and t and u are terms;
- $u = T(t_1,\ldots,t_n)$, where T is a n-ary function symbol in $\mathcal{F}$ and $u, t_1,\ldots,t_n$ are terms;
- $\mathbf{p}(t_1,\ldots,t_n)$, where $\mathbf{p}$ is a n-ary predicate symbol in $\mathcal{P}$ and $t_1,\ldots,t_n$ are terms.

The literals using concrete binary function (or predicate) symbols may be written in infix notation, as a syntactical convenience. A *triple pattern* is a class or property literal. We say that a triple $\boldsymbol{t}$ *matches* a triple pattern $\mathbf{p}$ iff:

- $\mathbf{p}$ is a class literal of the form $C(x)$, where x is a variable, and $\boldsymbol{t}$ is of the form (s, rdf:type, C);
- $\mathbf{p}$ is a property literal of the form $P(x,y)$, where x and y are variables and $\boldsymbol{t}$ is of the form (s, P, o).

Note that a triple does not match a literal of the forms $u = T(t_1,\ldots,t_n)$ or $\mathbf{p}(t_1,\ldots,t_n)$, where T is an n-ary function symbol in $\mathcal{F}$ and $\mathbf{p}$ is a n-ary predicate symbol in $\mathcal{P}$. A *rule body* B is a list of literals, separated by semi-colons. When necessary, we use "$B[x_1,\ldots,x_k]$" to indicate that the variables $x_1,\ldots,x_k$ occur in B. We say that B is *over* a vocabulary $\mathbf{V}$ iff all classes and properties that occur in B are from $\mathbf{V}$. As a notational convenience, a rule body B may include: 1) SPARQL *property paths* (shortly *paths*), either in prefix or in infix notation and 2) SPARQL *unary, binary* or *ternary operators*, either in prefix or in infix notation.

A *mapping rule from* $\mathbf{O_S}$=($\mathbf{V_S}$,$\Sigma_\mathbf{S}$) *to* $\mathbf{V_T}$, or simply a *rule from* $\mathbf{O_S}$=($\mathbf{V_S}$,$\Sigma_\mathbf{S}$) *to* $\mathbf{V_T}$, is an expression of one of the forms:

- $C(x) \leftarrow B[x]$, called a *class mapping*, where C is a class in $\mathbf{V_T}$ and $B[x]$ is a rule body over $\mathbf{V_S}$;
- $P(x,\ y) \leftarrow B[x,\ y]$, called a *property mapping*, where P is a property in $\mathbf{V_T}$ and $B[x,\ y]$ is a rule body over $\mathbf{V_S}$.

The expression on the left (right) of the arrow is called the *target* (*source*) *pattern* of the rule.

A *simple mapping* is a mapping rule of one of the forms:

- $C_\mathbf{T}(x) \leftarrow C_\mathbf{S}(x)$, where $C_\mathbf{T}$ is a class in $\mathbf{V_T}$ and $C_\mathbf{S}$ is a class in $\mathbf{V_S}$
- $P_\mathbf{T}(x, y) \leftarrow C_\mathbf{S}(x); P_\mathbf{S}(x, y)$, where $P_\mathbf{T}$ is a property in $\mathbf{V_T}$, $P_\mathbf{S}$ is a property in $\mathbf{V_S}$ and $C_\mathbf{S}$ is the domain of $P_\mathbf{S}$, defined in the source ontology $\mathbf{O_S}$.

3.2 Mapping Rules Patterns

This section proposes a set of MRs patterns that facilitate the design of RDF to RDF mappings by providing templates for MRs.

Each MRs pattern will represent a generic solution to a given RDF to RDF mapping problem. This section also proposes a more concise abstract syntax, based on MAs (Pequeno et al., 2016), for representing the MRs pattern. The MAs support most types of data restructuring that are commonly found when mapping RDF to RDF. Moreover, the MAs suffice to capture all types of mappings that can be expressed using the R2R language (Bizer and Schultz, 2010).

Let $\mathbf{O_S}$ = ($\mathbf{V_S}$, $\Sigma_\mathbf{S}$) be a source ontology and $\mathbf{O_T}$ = ($\mathbf{V_T}$, $\Sigma_\mathbf{T}$) be a target ontology. Assume that $\Sigma_\mathbf{S}$ and $\Sigma_\mathbf{T}$ both have constraints defining the domain and range of each property. Briefly, there are three types of MAs:

- Class Mapping Assertion (CMA), which is a class mapping;
- Object Property Mapping Assertion (OMA), which is a property mapping whose target predicate is an object property;
- Datatype Property Mapping Assertion (DMA), which is a property mapping whose target predicate is a datatype property.

Table 1 shows the formalism to express each type of CMA and some types of DMA, as well as the MRs patterns *induced* by these MAs. The whole formalism can be found in (Pequeno et al., 2016). In Table 1, we use the predicate $hasUri[A_1,\ldots,A_n](s,u)$ to generate unique URIs. Intuitively, let ψ be a CMA for a class $C_\mathbf{T}$ of $\mathbf{V_T}$ of form $C_\mathbf{T} \equiv C_\mathbf{S}[A_1,\ldots,A_n]$, $hasUri[A_1,\ldots,A_n](s,\ u)$ holds iff, when given a resource s of $C_\mathbf{S}$, u is the URI obtained by concatenating the namespace prefix for $C_\mathbf{T}$ and values of $A_1,\ldots,A_n$.

Table 2 shows examples of MAs that specify mappings between the source ontologies shown in Figs.1 and 2 and the target ontology in Fig. 3, as well as it shows the MRs induced by MAs in Table 2.

Consider, for example, the MRs ψ_1 and ψ_3. Those mapping rules specify that each triple s in *myspo:Genre* produces the following triples:

- (s *rdf:type* *mo:Genre*). (ψ_1)
- (s *ex:genreName* v), where v is a particular value of *dc:title*. (ψ_3)

4 MAPPING PATTERNS

The proposed patterns are addressed to designers of ontology-based on data exchange systems (mainly

Table 1: Transformation Rules Patterns.

MA	Mapping Assertion	Mapping Rules Pattern
CMA_1	ψ: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}$ / f - ψ is the name of CMA - $C_{\mathbf{S}}$ is a class name of $\mathbf{V_S}$ - f is an optional filter over instances of $C_{\mathbf{S}}$	$C_{\mathbf{T}}(s) \leftarrow C_{\mathbf{S}}(s)$; $f(s)$
CMA_2	ψ: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1,\ldots,A_n]$ / f - ψ is the name of CMA - $C_{\mathbf{S}}$ is a class name of $\mathbf{V_S}$ - $A_1,\ldots,A_n$ are datatype properties whose domain is $C_{\mathbf{S}}$ - f is an optional filter over instances of $C_{\mathbf{S}}$	$C_{\mathbf{T}}(u) \leftarrow C_{\mathbf{S}}(s)$; hasUri$[A_1,\ldots,A_n](s, u)$; $f(s)$
DMA_1	ψ: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi\ /P_{\mathbf{S}}$ / f/ T, where - ψ is the name of DMA - $P_{\mathbf{T}}$ is a datatype property whose domain is $C_{\mathbf{T}}$ (or a superclass of $C_{\mathbf{T}}$) - $P_{\mathbf{S}}$ is a datatype property whose domain is $C_{\mathbf{S}}$ (or a superclass of $C_{\mathbf{S}}$) - φ is an optional path from $C_{\mathbf{S}}$ to $C_{\mathbf{R}}$, where $C_{\mathbf{R}}$ is a class name of $\mathbf{V_S}$ - f is an optional filter over instances of $P_{\mathbf{S}}$ - T is an optional function that transforms values of datatype properties - There exists a CMA ψ that matches the domain D of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$	$P_{\mathbf{T}}(s,t) \leftarrow C_{\mathbf{S}}(s)$; $f(s)$; $\varphi(s,o)$; $P_{\mathbf{S}}(o,v)$; $f(v)$; $T(v,t)$
DMA_2	ψ: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1,\ldots,A_n]/A_i$, where - ψ is the name of DMA - $P_{\mathbf{T}}$ is a datatype property whose domain is $C_{\mathbf{T}}$ (or a superclass of $C_{\mathbf{T}}$) - A_i, $1 \leq$ i $\leq$n, are datatype properties whose domain is $C_{\mathbf{S}}$ - There exists a CMA ψ_D that matches the domain D of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}[A_1,\ldots,A_n]$	$P_{\mathbf{T}}(u,v) \leftarrow C_{\mathbf{S}}(s)$; $f(s)$; $A_i(s,v)$; hasUri$[A_1,\ldots,A_n](s, u)$

Table 2: Mapping Assertions.

Label	Mapping Assertion	Mapping Rules
ψ_1	*mo:Genre* ≡ *myspo:Genre*	*mo:Genre*(*s*) ← *myspo:Genre*(*s*)
ψ_2	*mo:Label* ≡ *myspo:MusicArtist*[*myspo:recordLabel*]	*mo:Label*(*u*) ← *myspo:MusicArtist*(*s*); *hasURI*[ψ_2](*s*,*u*)
ψ_3	*mo:Genre* / *ex:genreName* ≡ *myspo:Genre* / *dc:title*	*ex:genreName*(*s*,*t*) ← *myspo:Genre*(*s*); *dc:title*(*s*,*t*)
ψ_4	*mo:Label* / *ex:labelName* ≡ *myspo:MusicArtist*[*myspo:recordLabel*] / *myspo:recordLabel*	*ex:labelName*(*u*,*v*) ← *myspo:MusicArtist*(*s*); *myspo:recordLabel*(*s*,*v*) ; *hasURI*[ψ_2](*s*,*u*)
ψ_5	*mo:MusicArtist* ≡ *dbo:MusicalArtist*	*mo:MusicArtist*(*s*) ← *dbo:MusicalArtist*(*s*)

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

data integration ones) to better specify how the source ontologies are related to the target ontology and how to create mappings that transform source instances into target instances. We define various patterns in order to cover the more commons types of mapping found in the LOD (Ritze et al., 2009). Each pattern consists of a description of the goals of the pattern, the context, and constraints in which it can be used, a description of the solution in mapping formalisms and in R2R mapping language, examples and related patterns. The mapping formalisms abstract the way in which the correspondences and transformations of instance data between the ontologies can be specified and the R2R language provides an API that makes the mapping ready to be used in the practice. Other mapping languages can be used, instead of R2R one, for example, the SPARQL. If it is the case, we suggest the reader use our mapping formalism as a guide to creating the mappings in the SPARQL or other languages of your choice.

We have defined a mapping catalogue consisting of 12 mapping patterns. Figure 7 shows the classification of the mapping patterns in accordance with the type of mapping involved.

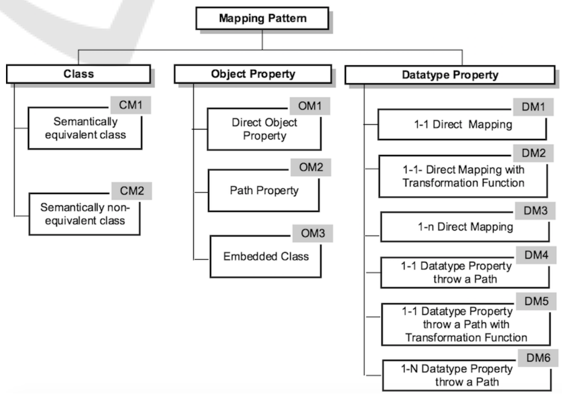

Figure 7: Pattern catalogue.

A mapping between ontology terms can be a mapping between classes or a mapping between properties. Since there are two types of properties (object and datatype), in Fig. 7, we grouped the patterns into three groups: *classes, object properties* and *datatype* *properties*. Each group was further divided in order to solve a specific mapping problem. For example, the class mapping patterns were divided into mapping patterns between semantically equivalent classes and mapping patterns between non-equivalent classes (determine when two classes are semantically equivalent or not is outside of the scope of this work, but the reader can see (Rodriguez and Egenhofer, 2003) for more details about this subject).

Due to limited space, in this paper, we exemplary illustrate one mapping pattern representative of the pattern catalogue. We refer to the reader to (Pequeno et al., 2016) for see the whole pattern catalogue.

In the remainder of this paper, consider $\mathbf{O_T}$=($\mathbf{V_T}$,$\Sigma_\mathbf{T}$) be a *target ontology*, with $\mathbf{V_T}$ and $\Sigma_\mathbf{T}$ being, respectively, the target vocabulary and the set of constraints of $\mathbf{O_T}$, and $\mathbf{O_S}$=($\mathbf{V_S}$,$\Sigma_\mathbf{S}$) be a *source ontology* with $\mathbf{V_S}$ and $\Sigma_\mathbf{S}$ being, respectively, the source vocabulary and the set of constraints of $\mathbf{O_S}$.

4.1 Mapping Pattern of Semantically Non-Equivalent Class

Name: Semantically Non-Equivalent Class Mapping
Alias: CM2
Problem: How should we specify the mapping of the instances of a class $C_\mathbf{S}$ in $\mathbf{V_S}$ into instances of a class $C_\mathbf{T}$ in $\mathbf{V_T}$?
Context:

- $C_\mathbf{T}$ and $C_\mathbf{S}$ are classes in vocabularies $\mathbf{V_T}$ and $\mathbf{V_S}$, respectively.
- $A_1,\ldots,A_n$ are datatype properties whose domain is $C_\mathbf{S}$.
- $C_\mathbf{T}$ and $C_\mathbf{S}$ are NOT semantically equivalent, i.e. they do not represent the same object of the real world
- f is a condition of selection (a predicate) over instances of $C_\mathbf{S}$ (f is optional).
- The terms may have the same name or different names in the different ontologies.

For example, the class *mo:Label* of **MyMusic** ontology corresponds to class/property combination *myspo:MusicArtist*[*myspo:recordLabel*] in **Myspace** ontology. It illustrates a mapping between not semantically equivalent classes (i.e., classes that do not represent the same object of the real world).
Force:

The mapping can be complete (when there is no condition of selection over instances, thus all instances of $C_\mathbf{S}$ are mapped into instances of $C_\mathbf{T}$) or partial (when only some instances of $C_\mathbf{S}$ are mapped, i.e., those instances that satisfy the condition of selection (filter)) .
Solution:

Mapping Rule: $C_\mathbf{T}(u) \leftarrow C_\mathbf{S}(s);\ f(s);$ hasUri[$A_1,\ldots,A_n$](s, u)

This rule specifies that for each triple < s rdf:type $C_\mathbf{S}$ >, such that $f(s)$ = true and u= hasUri[$A_1,\ldots,A_n$](s), a triple < u rdf:type $C_\mathbf{T}$ > is generated. Note that the instances of s and u have different URIs since they do NOT represent the same object in the real world. Thus, "$C_\mathbf{S}[A_1,\ldots,A_n]$" defines a "new class", whose instances are embedded in the instances of $C_\mathbf{S}$ defined by the above mapping rule. Therefore, the class $C_\mathbf{T}$ is semantically equivalent to the embedded class $C_\mathbf{S}[A_1,\ldots,A_n]$.

Mapping Assertion: ψ: $C_\mathbf{T} \equiv C_\mathbf{S}[A_1,\ldots,A_n]\ /\ f$ (CMA_2)

R2R mapping: Template T2

```
# Class Mappings
mp: ψC
    a r2r:ClassMapping ;
    r2r:prefixDefinitions "prefixExp" ;
    r2r:sourcePattern "?SUBJ a S:CS sQuery" ;
    r2r:targetPattern "?s a T:CT" ;
    r2r:transformation
        "?s=generateUri(?SUBJ, [A1,...,An])" .
```

S:$C_\mathbf{S}$ and T:$C_\mathbf{T}$ are directly obtained from the CMA_2. prefixExp and sQuery are variable with the same role than explained in CM1 pattern and generateUri() is the function to generate the new URI for $C_\mathbf{T}$ based on the predicate hasUri[$A_1,\ldots,A_n$] defined in the MR.

Example: Mapping between the non-semantically equivalent classes *mo:Label* and *myspo:MusicArtist*[*myspo:recordLabel*].

- **Mapping rule:** ψ_2 shown in Table 2
- **Mapping Assertion:** ψ_2 shown in Table 2
- **R2R mappings:**

```
# Class Mapping
mp:ψ2
    a r2r:ClassMapping ;
    r2r:prefixDefinitions "mo:... . myspo:...".
    r2r:sourcePattern "?SUBJ a
            myspo:MusicalArtist ;
            myspo:recordLabel ?s" ;
    r2r:targetPattern '?u a mo:Label";
    r2r:transformation "?u=
            generateUri(?SUBJ, [?s])" .
```

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

5 APPLYING MAPPING PATTERNS TO GENERATE R2R MAPPINGS

In this section, we present how to use the mapping patterns to generate the R2R mappings between a target ontology and a source one. In our proposal, the process to create R2R mappings to transform instances from an ontology into another one consists of two steps:

1. Define the MAs that formally specify the relationships between the target ontology and the source one.
2. Generate a set of R2R mappings based on the MAs generated in step 2, in order to populate the target ontology with values from the source(s) ontology(ies).

By using our MAs, the user focuses on the mapping itself, since: a) our language is less verbose than the usual mapping languages, and b) it is easy to learn. In the current work, the MAs are manually specified. However, we use the RBA tool (Vinuto, 2017), which has a GUI, to help us in this task.

The generation of the R2R mappings is based on the vocabulary of the ontologies (target and source) and on the MAs, which are part of the mapping patterns. Let $\mathcal{M}$ be a set of MAs that defines a mapping between the target ontology $\mathbf{O_T}$ and the source one $\mathbf{O_S}$ such that $\mathcal{M}$ satisfies the constraints identified to each mapping pattern that contains the MA. Figure 8 shows the algorithm to automatically generate the statements of R2R mappings from MAs in $\mathcal{M}$.

```
foreach class C_T in O_T do
    G_R2RclassMapping(C_T)
    foreach object property P_T whose domain is C_T do
        G_OMA_R2RpropMapping(P_T)
    end
    foreach datatype property P_T whose domain is C_T do
        G_DMA_R2RpropMapping(P_T)
    end
end
```

Figure 8: Generate the R2R mapping from MAs.

The algorithm 8 generates a set of R2R class mappings, at least one for each class $C_\mathbf{T}$ in $\mathbf{O_T}$, and a set of R2R property mappings, at least one for each property $P_\mathbf{T}$ in $\mathbf{O_T}$ since they have a MA specified. For each class $C_\mathbf{T}$ in $\mathbf{O_T}$, the algorithm first spans all CMAs of $C_\mathbf{T}$, in order to create the R2R class mappings through the algorithm *G_R2RclassMapping*. Then, it spans all datatype properties $P_\mathbf{T}$ whose domain is $C_\mathbf{T}$, in order to create R2R property mappings through the algorithm *G_DMA_R2RpropMapping*. Finally, the algorithm spans all object properties $P_\mathbf{T}$ whose domain is $C_\mathbf{T}$, in order to create R2R property mappings through the algorithm *G_OMA_R2RpropMapping*.

Figure 9 shows the procedure *G_R2RclassMapping*(). The R2R mapping is generated using the templates T1 or T2 in accordance with the type of mapping pattern (CMA1 or CMA2, respectively). Both templates use two variables that will be bound with values obtained from the CMA, and/or from the ontologies. These variables are: prefixExp, which keeps the prefixes presents in the elements of the CMA and sQuery, which keeps the expression used in the ***r2r:sourcePattern*** clause. Whether the CMA has a filter f, then the source pattern clause contains a filter expression. In this case, *FilterExp*() is used in order to convert f to the R2R syntax and its result is concatenated with the value of sQuery.

```
Procedure G_R2RclassMapping()
Input: C_T
// ψ_C is of form ψ_C:  C_T ≡ C_S/f or ψ_C or
// ψ_C is of form ψ_C: C_T ≡ C_S[A_1,...,A_n]/f
foreach CMA ψ_C of C_T do
    sQuery = NULL
    prefixExp = getPrefixes(ψ_C) // obtain prefixes given
        in ψ_C
    if f ≠ NULL then
        sQuery = sQuery + FilterExp(ψ_C)
    end
    if ψ_C is of form ψ_C: C_T ≡ C_S/f then
        // ψ_C is a CMA1 mapping pattern
        use template T1
    else
        // ψ_C is a CMA2 mapping pattern
        use template T2
    end
end
```

Figure 9: G_R2RclassMapping().

For example, the R2R mapping shown an example of the CMA2 mapping pattern was generated using ψ_2 and template T2. The ***r2r:prefixDefinitions*** clause was obtained from ψ_2 in addition with **MyMusic** ontology and **MySpace** ontology. The ***r2r:sourcePattern*** clause was obtained from the source pattern of the CMA ψ_2, being that **S:$C_\mathbf{S}$** = "*myspo:MusicalArtist*" and sQuery = "; myspo:recordLabel ?*s*". The ***r2r:targetPattern*** clause was obtained from the target pattern of the CMA ψ_2, being that **T:$C_\mathbf{T}$** = "*mo:Label*".

Algorithms *G_OMA_R2RpropMapping*() and *G_DMA_R2RpropMapping*() will not be described here due to space limitations. A detailed description of them can be found in (Pequeno et al., 2016).

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

6 RELATED WORK

In the literature, there are some works that propose patterns to deal with problems in data exchange scenarios. For example, (Ritze et al., 2009) propose patterns to detect correspondences between classes and properties; (Sváb-Zamazal et al., 2009) and (Scharffe et al., 2014) propose patterns to deal with the transformation of ontology to another (named the *ontology alignment* problem).

Defining correspondences between classes and properties is not the same as defining how an ontology can be filled from another. This means that it is not enough we identify the correspondences between the terms of different ontologies, we need to specify how exactly a resource can be added from other ontology(ies). In fact, in other scenarios (for example *schema mapping*) correspondences are used as the first step in approaches to load a schema based on data from other schemas. Therefore, none of these works presents a complete solution to the problem described in this paper.

In (Scharffe et al., 2014), for example, the authors focus on ontology mediation, which lies in the specification of correspondences between ontology terms. Their correspondences differ from our mappings since that they must be valid both side of the matching (in our mappings the assigned is from the source to the target only). Their solution is not enough to be used in our context because the authors do not really show how an instance of one ontology can be transformed in an instance of another. In (Rivero et al., 2012), the authors present RDF to RDF mapping patterns (the same context of our patterns), however, different of our work, their paper only shows the definition of the problem included in each pattern and some examples, none solution is discussed to resolve them.

7 CONCLUSION

This paper presented a proposal to semi-automatically generate R2R mappings using mapping patterns. The solution presented in the pattern allows the users not only specify mappings between terms of two ontologies in a clear and concise way but provides a mapping that is ready to be used in the real scenarios. Although mapping patterns seem to be complex for a normal user, they group the most common problems that a designer must encounter when he/she is defining mappings between ontologies. Because mapping problems are catalogued and there are examples in each mapping pattern, it is easier to identify and find a solution for each situation.

We have implemented a tool, named RBA (Vinuto, 2017), for helping the designer in the process of definition of the mappings, which uses the proposed patterns. We have tested our approach in some toy examples, but we intend to make some experiments for measure the time needed for the creation of the mapping patterns in order to determine whether the effort in creating the MAs is higher than the creation of the actual R2R mappings.

As a future work, we intend to carry out a deep study to show how our proposal is generally useful in different use cases of R2R and to carry out a detailed comparison with other state-of-the-art tools.

REFERENCES

Bizer, C. and Schultz, A. (2010). The R2R framework: Publishing and discovering mappings on the web. In *COLD'10*, volume 665.

Lutz, C. (2002). Description logics with concrete domains—a survey. In *AiML'02)*, France.

Pequeno, V., Vidal, V., Vinuto, T., and Galhardas, H. (2015). Automatic generation of R2R mappings from correspondence assertions. In *SBBD*.

Pequeno, V. M., Vidal, V. M. P., and Vinuto, T. (2016). Towards automatic generation of R2R mappings. Technical report, INESC-ID.

Ritze, D., Meilicke, C., Šváb-Zamazal, O., and Stuckenschmidt, H. (2009). A pattern-based ontology matching approach for detecting complex correspondences. In *OM'09*, pages 25–36, Germany.

Rivero, C. R., Schultz, A., Bizer, C., and Ruiz, D. (2012). Benchmarking the performance of linked data translation system. In *LDOW'12*, France.

Rodriguez, M. A. and Egenhofer, M. J. (2003). Determining semantic similarity among entity classes from different ontologies. *TKDE'03*, 15(2):442–456.

Scharffe, F., Zamazal, O., and Fensel, D. (2014). Ontology alignment design patterns. *Knowl. Inf. Syst.*, 40(1):1–28.

Schultz, A., Matteini, A., Isele, R., Bizer, C., and Becker, C. (2011). LDIF - Linked Data Integration Framework. In *COLD'11*.

Shvaiko, P. and Euzenat, J. (2013). Ontology matching: State of the art and future challenges. *TKDE'13*, 25(1):158–176.

Sváb-Zamazal, O., Svátek, V., and Scharffe, F. (2009). Pattern-based ontology transformation service. In *KEOD'09*, pages 210–223, Portugal.

Vinuto, T. d. S. (2017). Uma abordagem para a geração semiautomática de mapeamentos R2R baseado em padrões. Master's thesis, Universidade Federal do Ceará, Fortaleza, Brazil.