— INESC-ID Lisboa Tech. Rep. 7/2016 —

Valéria Pequeno[1] and Vânia M. P. Vidal[2] and Tiago Vinuto[2]
[1]Av. Prof. Doutor Aníbal Cavaco Silva 2744-016 Porto Salvo, Portugal
INESC-ID
[2] Campus do Pici - Bloco 710 60455-900 Fortaleza, CE, Brazil
Universidade Federal do Ceará

This revision: October 20, 2016

# Towards Automatic Generation of R2R Mappings

Valéria Pequeno[1] and Vânia M. P. Vidal[2] and Tiago Vinuto[2]

[1] INESC-ID
IST Tagus Park, Av Prof Cavaco Silva 2780-990 Porto Salvo, Portugal
vmp@inesc-id-pt
[2] UFC
Campus do Pici - Bloco 710 60455-900 Fortaleza, CE, Brazil
{vvidal, tiagosv}@lia.ufc.br

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. We propose the use of mapping patterns to automatically generate R2R mappings between RDF vocabularies. In this paper, we first formally specify the mapping formalism to transform data from a source ontology to a target ontology vocabulary. Second, we introduce the proposed mapping patterns. Third, we present a method to automatically generate R2R mappings using the mapping patterns. Finally, we present a sketch of a tool for helping the designer in the process of generating R2R mapping.

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

## 1 Introduction

Nowadays, there is a large number of datasets (of 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 accessibility of this global data space creates new opportunities for people (and machines) to reuse the data in different contexts and applications. For example, biomedical data relating to experiences (with its associated literature) can be integrated into **DBpedia**[3]. This enables a better classification of concepts and their description in several languages (thanks to **DBpedia**). The use of Linked Data is also a promising solution for data integration, since there is a standardized data model (RDF[4]) and a query language (SPARQL[5]) that recognize the relationships between the vocabularies used by sets of Linked 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** and **Music ontology**[6] use

[3] http://dbpedia.org/resource/
[4] https://www.w3.org/RDF/
[5] http://www.w3.org/TR/rdf-sparql-query/
[6] http://musicontology.com/

their own proprietary vocabularies to represent data about musical artists. The resulting data heterogeneity is a major obstacle to build useful Linked Data applications and thus to perform the promise of the Web of Linked Data: to set up applications to work on top of a single global data-space which enables them to discover and integrate new data sources at run-time.

Translating data from Linked Data sources (*the source ontologies*) to the vocabulary that is expected by a linked data application (*the target ontology*) requires a large number of mappings. **RDFS!**[7] and **OWL!**[8] languages and SKOS vocabulary[9] provide the terms *rdfs:subClassOf* / *rdfs:subPropertyOf*, *owl:equivalentClass* / *owl:equivalentProperty*, as well as the mapping of properties of the SKOS vocabulary to publish mappings between RDF vocabularies. However, data translation between the ontologies usually requires lots of structural transformations as well as complex property value transformations using various, possibly, functions and these type of mapping cannot be performed by using only **RDFS!** (**RDFS!**), **OWL!** (**OWL!**) and **SKOS!** (**SKOS!**) terms. 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, there is not a specific editor to define R2R mappings in order to assist the user in this process. In addition, the R2R framework[10] only has general error messages that, again, 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 and itonly provides the language to define the mappings. Besides, the definition of the R2R mappings is error-prone and time consuming (mainly when complex mappings are required). It calls for the development of methods and tools to support the deployment of mappings using R2R.

As main contribution of this paper, we suggest an automated pattern based approach to generate 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 automatic generation of R2R mappings. Other contributions of this paper are: (i) the formal definition of Mapping Assertions (MAs) (informally addressed in [Pequeno et al., 2015]) as a convenient way to manually specify mappings between RDF vocabularies; and (ii) the sketch of a tool for helping the designer in the definition of the MAs and in the generation of R2R mappings.

The rest of the paper is organized as follows. In Section 2, we introduce our formalism to define mappings. In Section 3, we show a motivating example. In Section 4, we briefly present the R2R mapping language. In Section 5, we present our proposal for generating R2R mappings using mapping patterns. In Section 6, we point out how automatically to generate R2R mappings by applying mapping patterns. In Section 7, we show a sketch of our tool for helping the designer in the definition of the MAs and the automatic generation of R2R mappings. In Section 8, we discuss about the related work. Finally, in Section 9, we present our conclusions.

[7] https://www.w3.org/TR/rdf-schema/
[8] https://www.w3.org/OWL/
[9] https://www.w3.org/TR/skos-reference/#mapping
[10] http://r2r.wbsg.de/

## 2 Mapping Representation

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[11], 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. Some examples of mapping rules are presented later.

A *vocabulary* $\mathbf{V}$ is a set of *classes* and *properties*. An *ontology* is a pair $\mathbf{O}=(\mathbf{V},\Sigma)$ such that $\mathbf{V}$ is a vocabulary and $\Sigma$ 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[12] 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 $\mathbf{V_S}$.

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) some of the SPARQL *property paths*, either in prefix or in infix notation and 2) some of the SPARQL *unary, binary* or *trinary*

[11] https://www.w3.org/Submission/SWRL/
[12] Internationalized Resource Identifier.

Table 1: Allowed Path Expressions.

| **Property Path Type** | **Notation** | **Translation** |
|---|---|---|
| *Inverse path* | $P\hat{}(t_1, t_2)$ | $P(t_2, t_1)$ |
| | $t_1\ P\hat{}\ t_2$ | |
| *Sequence path* | $P_1/P_2/\ldots P_k(t_1, t_2)$ | $P_1(t_1, x_2)$; $P_2(x_2, x_3)$; $\ldots$ $P_k(x_k, t_2)$ |
| | $t_1\ P_1/P_2/\ldots P_k\ t_2$ | |
| *Fixed Length path* ($k \geq 0$) | $P\{k\}(t_1, t_2)$ | $P(t_1, x_2)$; $P(x_2, x_3)$; $\ldots$ $P(x_k, t_2)$ ($P$ repeated k times) |
| | $t_1\ P\{k\}\ t_2$ | |

*operators*, either in prefix or in infix notation. Table 1 lists the allowed property paths and their translations, where $P, P_1, P_2, \ldots, P_k$ are properties and $x_2, x_3, \ldots, x_k$ are variables not occurring in the rule body $B$. Table 2 exemplifies SPARQL operators, where $x, x_1$ and $x_2$ are variables occurring in the rule body $B$.

Table 2: Examples of Allowed SPARQL Operators.

| **SPARQL operator** | **Meaning** |
|---|---|
| $x_1 \| \| x_2$ | logical or |
| $x_1 \& \& x_2$ | logical and |
| bound($x$) | returns true if $x$ is bound to a value |
| $!x$ | not $x$ |

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 $\mathbf{O_S}$.

We assume that the rules are not mutually recursive. Hence, they act as definitions of concepts in $\mathbf{V_T}$ in terms of concepts in $\mathbf{V_S}$. Indeed, we consider a mapping rule $\boldsymbol{r}$ as a shorthand notation the first-order sentence:

- $\forall x_1 \ldots \forall x_k(C(x_1) \Leftrightarrow B[x_1, \ldots, x_k])$, if $\boldsymbol{r}$ is of the form $C(x_1) \leftarrow B[x_1, \ldots, x_k]$, where $x_1, \ldots, x_k$ are the variables that occur in $B$;

– $\forall x_1 \ldots \forall x_k(P(x_1, x_2) \Leftrightarrow B[x_1, \ldots, x_k])$, if $\boldsymbol{r}$ is of the form $P(x_1, x_2) \leftarrow Bx_1, \ldots, x_k]$, where $x_1, \ldots, x_k$ are the variables that occur in $B$.

Therefore, the mapping rules have a first-order semantics. In particular, we stress that the concrete function and predicate function symbols have a fixed interpretation.

In this work, we focus on some of the most common type of mapping rules.

## 3 Motivating Example

Let us consider as an example the mapping between **MyMusic** ontology (the target) and both sources **DBpedia**[13] and **MySpace**[14]. **DBpedia** (*dbo, dbp*) and **MySpace** (*myspo*) ontologies are shown, respectively, in Figs. 1 and 2, while **MyMusic** is shown in Fig. 3. All are represented using the UML Class Diagram notation.

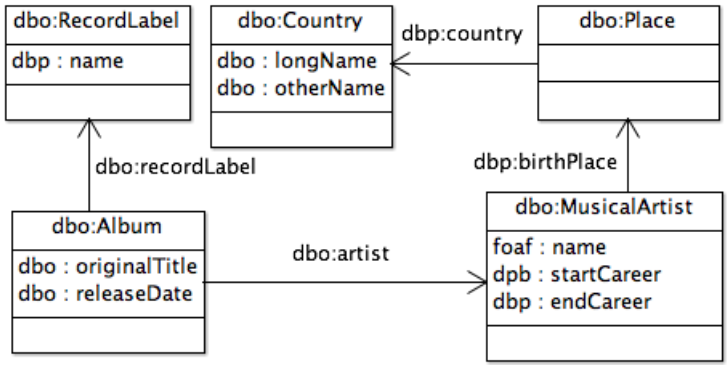


Fig. 1: A simplified fragment of the **DBpedia** ontology.

The fragment of the **DBpedia** (shown in Fig. 1) depicts information about artists and related aspects. **MySpace** shown in Fig. 2 provides part of a Resource Description Framework (RDF) representation of MySpace users. **MyMusic** reuses terms from four well-known vocabularies: FOAF (**Friend of a friend**)[15], MO (**Music ontology**)[16], DC (**Dublin core**)[17] and **VCARD**[18]. We use the prefix "moa" for new terms defined in the **MyMusic** ontology. For example, *moa:labelName* keeps the name of the record label and *moa:careerDuration* keeps the data at which the activity of the artist starts and end.

Table 3 shows some examples of mapping rules for our running example. The mapping rule $R1$ maps *mo:Record* (of the target) to *dbo:Album* (of the source). It indicates that each triple of *dbo:Album*, such that the predicate *dbo:releaseDate* $>$ '2013-01-01' is satisfied, produces a

[13] http://dbpedia.org/ontology/ and http://dbpedia.org/property/.
[14] http://purl.org/ontology/myspace/
[15] http://xmlns.com/foaf/0.1/
[16] http://purl.org/ontology/mo/
[17] http://purl.org/dc/elements/1.1/
[18] http://www.w3.org/2006/vcard/ns#

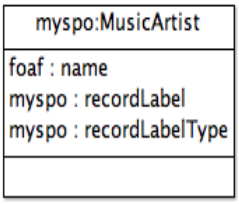


Fig. 2: A simplified fragment of the **MySpace** ontology.

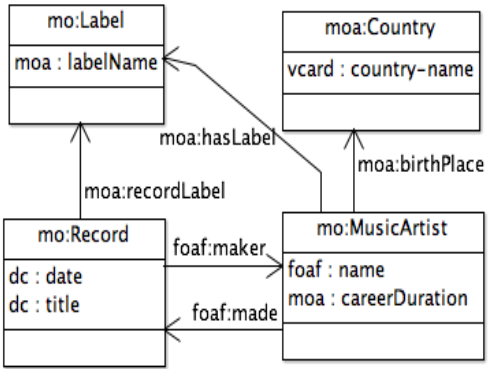


Fig. 3: A simplified fragment of the **MyMusic** ontology.

triple of *mo:Record*. The rule $R2$ maps *dc:title*, whose domain is *mo:Record*, to *dbo:originalTitle*, whose domain is *dbo:Album*. It indicates that each triple of *dbo:originalTitle* in *dbo:Album* produces a triple of *dc:title* whose object triple is the value of *dbo:originalTitle*.

Table 3: Some examples of mapping rules.

| **ID** | **Mapping Rule** |
|---|---|
| $R1$ | *mo:Record*($s$) $\leftarrow$ *dbo:Album*($s$) ; *dbo:releaseDate*($s$, $v$) $>$ '2013-01-01' |
| $R2$ | *dc:title*($s$, $v$) $\leftarrow$ *dbo:Album*($s$) ; *dbo:originalTitle*($s$, $v$) |

Table 4 shows more complex examples of mapping rules. The mapping rule $R3$ maps *mo:Label* with *myspo:recordLabel*. It indicates that each triple $\langle s$ *myspo:recordLabel* $w\rangle$, such that the predicate *myspo:recordLabelType*='Major' is satisfied, produces a triple $\langle u$ rdf:type *mo:Label*$\rangle$, where $u$ is obtained by using the functions concat() and xpath:encode-for-uri(). The function xpath:encode-for-uri() returns the string argument with certain characters escaped to enable the resulting string to be used as a path segment in a URI. The function concat() is used here to generate the new URI $u$ by concatenating the subject and the object triple of *myspo:recordLabel*. This ensure that the URIs of the triples produced by rule $R3$ are unique. Mapping rule $R4$ maps *moa:labelName*, whose domain is *mo:Label*, with *myspo:recordLabel*, whose domain is *myspo:MusicArtist*. It indicates that each triple $\langle s$ *myspo:recordLabel* $w\rangle$, such that the predicate *myspo:recordLabelType*='Major' is satisfied, produces a triple $\langle u$ *moa:labelName* $w\rangle$, where $u$ is obtained like rule $R3$.

Table 4: Some more complex examples of mapping rules.

| ID | Mapping Rule |
|---|---|
| $R3$ | *mo:Label*($u$) ← *myspo:MusicArtist*($s$) ; *myspo:recordLabel*($s$,$w$) ;<br>concat($s$, xpath:encode-for-uri($w$), $u$) ; *myspo:recordLabelType*($s$, $v$) = 'Major' |
| $R4$ | *moa:labelName*($u$, $w$) ← *myspo:MusicArtist*($s$) ; *myspo:recordLabel*($s$,$w$) ;<br>concat($s$, xpath:encode-for-uri($w$), $u$) ; *myspo:recordLabelType*($s$, $v$) = 'Major' |

# 4 R2R Mapping Language

R2R[19] 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 its 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 right hand side of our mapping rule. The target pattern is used to produce triples in the target vocabulary. It corresponds to the left hand side of our mapping rule. A R2R mapping may consist of multiple target patterns but has a single source pattern. R2R mappings also have, optionally, a clause **r2r:transformation** that defines how the values in **r2r:targetPattern** are transformed and a clause **r2r:mappingRef** that refers to a **r2r:classMapping** defined before. Also is optional the clause **r2r:prefixDefinitions**, used to abbreviate URIs inside the **r2r:sourcePattern** or **r2r:targetPattern**.

Figure 4 shows the R2R mapping between *mo:Label* and *myspo:recordLabel* (the same mapping that was defined using the mapping rule - R3 in Table 4). The instance variable ?SUBJ must be used in every source pattern and is reserved for representing the instances that are the focus of the mapping. Lines 04-05 in Fig. 4 specifies how to obtain all *myspo:recordLabel* resources of *myspo:MusicArtist* whose predicate *myspo:recordLabelType* = "Major" is true. Line 06 specifies the format of the target triple (i.e., the triple of *mo:Label*). When mapping *mo:Label* with *recordLabel*, a new URI must be generated based on the URI of *myspo:recordLabel* and on the property value, in order to guarantee that the new URI is unique. Thus, the subject triple of the *mo:Label* is the new URI generated in Line 07. The new URI $u$ is obtained by using the functions concat() and xpath:encode-for-uri() (already explained).

Figure 5 shows a R2R mapping between *moa:labelName* and *myspo:recordLabel* (the same mapping that was defined using the mapping rule - R4 in Table 4). It specifies that each triple <?SUBJ *myspo:recordLabel* ?r> in *myspo:MusicArtist*, such that the predicate *myspo:recordLabelType* = "Major" is true, produces a triple <?$u$ *moa:labelName* ?r> (lines 04-07). The **r2r:mappingRef** makes references to the class mapping mp:Label_to_recordLabel shown in Fig. 4. It is used mainly to reduce redundancy: the source pattern of the mp:Label_to_record-

[19] http://wifo5-03.informatik.uni-mannheim.de/bizer/r2r/spec/

```
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.                      myspo:recordLabelType ?t . FILTER(?t = 'Major')";
06.   r2r:targetPattern "?u a mo:Label" ;
07.   r2r:transformation "?u= concat(?SUBJ, xpath-encode-for-uri(?r))" .
```

Fig. 4: R2R mapping between *mo:Label* and *myspo:recordLabel*.

Label is jointed with the source pattern of mp: labelName_to_recordLabel. Also in this mapping, we need to generate a URI, which must be the same generated in the mapping shown in Fig. 4.

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

Fig. 5: R2R mapping between *moa:labelName* and *myspo:recordLabel*.

Complex mappings, such as these presented in Figures 4 and 5, require expertise on the involved ontologies, as well as on the R2R language used to define the mappings. This task can be tedious and error-prone. A library of mapping patterns will facilitate the generation of R2R mappings by providing templates modeling complex mappings such as the ones give above.

## 5 Mapping Patterns

Mapping patterns are generic solutions to solve a given mapping problem [Šváb, 2007]. In this article, as proposed in [Scharffe et al., 2014], we consider *ontology mapping* as a design problem, where each mapping rule, between source and target ontologies, needs to be defined. This task can be long and tedious if the ontologies are large and in presence of structural heterogeneity, since state of the art mapping algorithms only can automatically discover simple mappings.

We propose to use design patterns to help the definition of mapping rules [Scharffe et al., 2014]. Each mapping pattern will represent a generic solution to a given mapping problem. In this paper, we propose a set of ontology mapping patterns that facilitates ontology mapping design by providing templates for modeling mapping rules. The design pattern also provides the

template for the R2R mapping to implement the mapping rule. The patterns can represent all types of ontology mismatches problems found in the literature [Klein, 2001]. We also propose a more concise abstract syntax, called *Mapping Assertion (MA)*, for representing the mapping rules associated with each mapping patterns.

## 5.1 Mapping Pattern Template

A pattern template enables to represent patterns in a standard way. They provide a simple format that can be used to describe a pattern. Our mapping pattern use classical elements from design patterns literature [Scharffe et al., 2014]. In our approach, a mapping pattern is formed by the following basic elements:

| | |
|---|---|
| Name: | **Name**: the name of the pattern |
| | **Alias**: alternative names or synonyms for the pattern |
| Problem: | **Problem**: a description of the goals of the pattern |
| | **Context**: the applicability of the pattern |
| Solution: | Description of the solution using mapping formalisms: |
| | **Mapping Assertions**, |
| | **Mapping rules** and |
| | **R2R Mappings** |

Examples of mapping patterns are shown in Section 5.3.

## 5.2 Mapping Assertions

Mapping Assertion (MA) is a formal and declarative language used to specify a subset of mapping rules in a high-level abstraction. There are proposals to define MAs in various data models (before named as Correspondence Assertions (CAs), e.g., XML [Vidal et al., 2001] and from relational to RDF [Vidal et al., 2013]). Here, MAs are used to specify some types of mappings between RDF data models, in a more concise syntax than other existent approaches.

There are three types of MAs:

- Class Mapping Assertion (CMA), which is a class mapping (see Section 2);
- 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.

MAs are formally defined as follows. Before that, we present the concept of *embedded class*, which is used in the definitions of some MAs.

**Definition 1 (Embedded Class).** *Let $C$ be a class in a vocabulary $\mathbf{V}$. Let also $A_1, \ldots, A_n$ be datatype properties whose domain is $C$. We said $C' = C[A_1, \ldots, A_n]$ is an embedded class of $C$ iff each instance of $C'$ is uniquely identified by the datatype properties $A_1, \ldots, A_n$, that is, no two distinct instances of $C'$ can coincide on the values of all properties $A_i$.* □

**Definition 2 (CMA).** *Let $C_T$ and $C_S$ be classes in $V_T$ and $V_S$, respectively. Let also $C[A_1, \ldots, A_n]$ be an embedded class of $C_S$. A Class Mapping Assertion (CMA) is an expression of one of the following forms:*

1. $\psi$: $C_T \equiv C_S \,/\, f$
2. $\psi$: $C_T \equiv C_S[A_1, \ldots, A_n] \,/\, f$

*where $\psi$ is the name of the CMA, $f$ is a selection predicate (filter) over (instances of) $C_S$ and $f$ is optional.* □

Table 5 shows examples of CMAs. $\psi_1$ specifies that each instance of the class *dbo:MusicalArtist* produces an instance to the class *mo:MusicArtist*. $\psi_2$ specifies that each instance of the class *dbo:Album*, such that the selection predicate *dbo:releaseDate* $>$ "2013-01-01" is true, produces an instance to the class *mo:Record*. $\psi_3$ specifies the mapping between *mo:Label* and *myspo:recordLabel* shown in Section 3. In $\psi_3$, we said that *myspo:MusicArtist*[*myspo:recordLabel*] is an embedded class of class *myspo:MusicArtist*.

Table 5: Examples of CMAs.

| |
|---|
| $\psi_1$: *mo:MusicArtist* $\equiv$ *dbo:MusicalArtist* |
| $\psi_2$: *mo:Record* $\equiv$ *dbo:Album* / *dbo:releaseDate*$>$ "2013-01-01" |
| $\psi_3$: *mo:Label* $\equiv$ *myspo:MusicArtist*[*myspo:recordLabel*] |

**Definition 3 (OMA).** *Let $C_T$ be a class in $V_T$ and $C_S$ and $C_R$ be classes in $V_S$. Let $P_T$ be an object property whose domain is $C_T$ (or a superclass of $C_T$) and $P_S$ be an object property whose domain is $C_S$ (or a superclass of $C_S$). Let also $C[A_1, \ldots, A_n]$ be an embedded class of $C_S$. An Object Property Mapping Assertion (OMA) is an expression of one of the following forms:*

1. $\psi$: $C_T/P_T \equiv C_S[A_1, \ldots, A_n]/NULL$
2. $\psi$: $C_T/P_T \equiv C_S/P_S/f$
3. $\psi$: $C_T/P_T \equiv C_S/\varphi/f$

*where $\psi$ is the name of the OMA, $\varphi$ is a path from $C_S$ to $C_R$, $f$ is a selection predicate (filter) over (instances of) $C_R$ and $f$ is optional.* □

Table 6 shows an example of OMA. $\psi_4$ specifies that each path formed by the RDF triples obeying the triple patterns: $<?s$ *dbp:birthPlace* $?u>$ $<?u$ *dbp:country* $?t>$ produces a RDF triple with the triple pattern: $<?s$ *moa:birthPlace* $?t>$.

**Definition 4 (DMA).** *Let $C_T$ be a class in $V_T$ and $C_S$ and $C_R$ be classes in $V_S$. Let $P_T$ be a datatype property whose domain is $C_T$ (or a superclass of $C_T$) and $P_S, P_{S_1}, \ldots, P_{S_n}$ be datatype*

Table 6: Example of an OMA.

| $\psi_4$:*mo:MusicArtist* / *moa:birthPlace* $\equiv$ *dbo:MusicalArtist* / [*dbp:birthPlace*/*dbp:country*] |
|---|

*properties whose domain is $C_S$ (or a superclass of $C_S$). Let $P_R$, $P_{R_1}, \ldots, P_{R_m}$ be datatype properties whose domain is $C_R$. Let also $C[A_1, \ldots, A_n]$ be an embedded class of $C_S$. A Datatype Property Mapping Assertion (DMA) is an expression of one of the following forms:*

1. $\psi$: $C_T/P_T \equiv C_S/P_S/f/T$
2. $\psi$: $C_T/P_T \equiv C_S/\varphi/P_R/f/T$
3. $\psi$: $C_T/P_T \equiv C_S/\{P_{S_1}, \ldots, P_{S_n}\}/T$
4. $\psi$: $C_T/P_T \equiv C_S/\varphi/\{P_{R_1}, \ldots, P_{R_m}\}/T$
5. $\psi$: $C_T/P_T \equiv C_S[A_1, \ldots, A_n]/A_i$ $(1 \leq i \leq n)$

*where $\psi$ is the name of the DMA, $\varphi$ is a path from $C_S$ to $C_R$, $f$ is a selection predicate (filter) over (instances of) $C_S$ (item 1 and 3) or $C_R$ (item 2 and 4), $T$ is a transformation function, and $f$ and $T$ are optional.* □

Table 7 shows examples of DMAs. $\psi_5$ specifies that each RDF triple obeying the triple pattern: $<?s$ *dbo:originalTitle* $?t>$ produces a RDF triple with the triple pattern: $<?s$ *dc:title* $?t>$. $\psi_6$ specifies that each RDF triple obeying the triple pattern: $<?s$ *myspo:recordLabel* $?n >$ produces a triple with the triple pattern: $<?t$ *moa:labelName* $?n >$, where $?t$ is the URI generated for the embedded class *myspo:MusicalArtist*[*myspo:recordLabel*] using the instance $?s$. $\psi_7$ specifies that each set of RDF triples obeying the triple patterns: $<?s$ *dbp:startCareer* $?v >$ and $<?s$ *dbp:endCareer* $?w >$ produces a triple with the triple pattern: $<?t$ *moa:careerDuration* $?n >$, where $?n$ is the result of the concatenation of ?*v* and ?*w*.

Table 7: Examples of DMAs.

| |
|---|
| $\psi_5$: *mo:Record /dc:title* $\equiv$ *dbo:Album /dbo:originalTitle* |
| $\psi_6$: *mo:Label* / *moa:labelName* $\equiv$ *myspo:MusicArtist*[*myspo:recordLabel*] / *myspo:recordLabel* |
| $\psi_7$: *mo:MusicArtist* / *moa:careerDuration* $\equiv$ *dbo:MusicalArtist* / {*dbp:startCareer*, *dbp:endCareer*} |

We said that a MA is simple when it has one of the following forms: $\psi$: $C_T \equiv C_S$ or $\psi$: $C_T/P_T \equiv C_S/P_S$, where $P_T$ and $P_S$ are datatype properties or object properties. Otherwise, MA is complex. In Tables 5, 6 and 7, the MAs $\psi_1$ and $\psi_5$ are examples of simple MAs; $\psi_2$, $\psi_3$, $\psi_4$, $\psi_6$ and $\psi_7$ are examples of complex MAs.

For each MA, there is a respective mapping rule that contains the semantics of the MA. Let $\mathcal{M}$ be a set of MAs, $C_{\mathbf{T}}$ be a class in $\mathbf{V_T}$, $C_{\mathbf{S}}$ and $C_{\mathbf{R}}$ be classes in $\mathbf{V_S}$ and $C_{\mathbf{S}}[A_1,\ldots,A_n]$ be an embedded class of $C_{\mathbf{S}}$ . Let also $P_{\mathbf{T}}$ be a property whose domain is $C_{\mathbf{T}}$ (or a superclass of $C_{\mathbf{T}}$) and $P_{\mathbf{S}}$ be a property whose domain is $C_{\mathbf{S}}$ (or a superclass of $C_{\mathbf{S}}$). Table 8 shows the MAs and respective mapping rules. The symbols in grey are optional ones, that is, they can or cannot appear in the MAs/mapping rules.

Table 8: MAs and Mapping rules

| **Mapping Assertion** | **Mapping Rule** |
|---|---|
| $\psi$: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}$ / $f$ | $C_{\mathbf{T}}(s) \leftarrow C_{\mathbf{S}}(s)$ ; $f(s)$ |
| $\psi$: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1,\ldots,A_n]$ / $f$ | $C_{\mathbf{T}}(u) \leftarrow C_{\mathbf{S}}(s)$ ; GenerateURI[$\psi$]($s$,$u$) ; $f(s)$ |
| $\psi$: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/P_{\mathbf{S}}$ / $f$/ $\mathcal{T}$, where<br>-$P_{\mathbf{T}}$ and $P_{\mathbf{S}}$ are datatype properties<br>- $\mathcal{M}$ has a CMA $\psi$ that matches the domain $D$ of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$ | $P_{\mathbf{T}}(s,t) \leftarrow C_{\mathbf{S}}(s)$; $f(s)$; $\varphi(s,o)$ ;<br>$P_{\mathbf{S}}(o,v)$; $f(v)$; $\mathcal{T}(v,t)$ |
| $\psi$: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1,\ldots,A_n]/A_i$, where<br>- $P_{\mathbf{T}}$ and $A_i$ are datatype properties<br>- $\mathcal{M}$ has a CMA $\psi_D$ that matches the domain $D$ of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}[A_1,\ldots,A_n]$ | $P_{\mathbf{T}}(u,v) \leftarrow C_{\mathbf{S}}(s)$ ; $f(s)$ ; $A_i(s,v)$ ;<br>GenerateURI[$\psi_D$]($s$,$u$) |
| $\psi$: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/\{P_{\mathbf{R}_1},\ldots,P_{\mathbf{R}_n}\}/T$, where<br>- $\mathcal{M}$ has a CMA $\psi_D$ that matches the domain $D$ of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$<br>If exists a $\varphi$ such that $\varphi$ is a path of $C_{\mathbf{S}}$ $to C_{\mathbf{R}}$, then $\mathcal{M}$ has a CMA $\psi_N$ that matches the range N of $P_{\mathbf{T}}$ with $C_{\mathbf{R}}$ | $P_{\mathbf{T}}(s,w) \leftarrow C_{\mathbf{S}}(s)$ ; $f(s)$ ; $\varphi(s,o)$ ;<br>$P_1(o,v_1);\ldots;P_n(o,v_n)$ ;<br>concat($v_1,\ldots,v_n,v$); $T(v,w)$ |
| $\psi$: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1,\ldots,A_n]/NULL$, where<br>- $\mathcal{M}$ has a CMA $\psi_D$ that matches the domain $D$ of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$<br>- $\mathcal{M}$ has a CMA $\psi_N$ that matches the range $N$ of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$ | $P_{\mathbf{T}}(s,u) \leftarrow C_{\mathbf{S}}(s)$; $f(s)$;<br>GenerateURI[$\psi_D$]($s$,$u$) |
| $\psi$: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/P_{\mathbf{S}}$ / $f$, where<br>-$P_{\mathbf{T}}$ and $P_{\mathbf{S}}$ are object properties<br>- $\mathcal{M}$ has a CMA $\psi_D$ that matches the domain $D$ of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$<br>- $\mathcal{M}$ has a CMA $\psi_N$ that matches the range $N$ of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$<br>- if exists a $\varphi$ such that $\varphi$ is a path of $C_{\mathbf{S}}$ to $C_{\mathbf{R}}$, then $\mathcal{M}$ has a CMA $\psi_N$ that matches the range $N$ of $P_{\mathbf{T}}$ with $C_{\mathbf{R}}$ | $P_{\mathbf{T}}(s,v) \leftarrow C_{\mathbf{S}}(s)$ ; $f(s)$ ; $\varphi(s,o)$ ;<br>$P_{\mathbf{S}}(o,v)$ ; $f(v)$ |

In Table 8, we use the function *GenerateUri*[$\psi$]() instead of the functions concat() and xpath:encode-for-uri() (shown in Table 4) as a way to simplify and standardize the generation of new URIs. Intuitively, let $\psi$ be a CMA for a class $C_{\mathbf{T}}$ of $\mathbf{V_T}$ of form $C_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1,\ldots,A_n]$, *GenerateUri*[$\psi$]($s$,$u$), 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 $A_1,\ldots,A_n$.

## 5.3 Pattern Library

Until now, we give the overview of the proposed patterns to map ontologies using mapping rules, MA and R2R mappings. We developed a pattern library containing the most common types of mappings. Figure 6 shows the classification of the mapping patterns in accordance with the type of mapping involved.

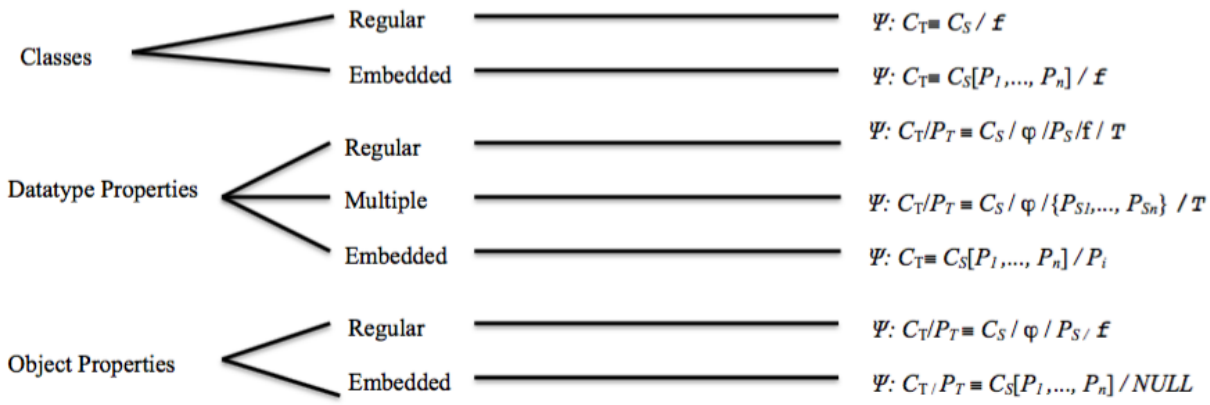


Fig. 6: Pattern Library.

In Fig. 6, we group all similar cases of MAs in a single structure. The result was a taxonomy containing 3 groups (one for each kind of MA) and 7 sub-groups. CMAs are divided into regular and embedded, to distinguish mappings between classes and mappings of classes to properties inside another class. DMAs are divided into regular, multiple and embedded, to distinguish mappings between single datatype properties, mappings of datatype properties to various datatype properties and mappings of datatype properties to properties of an embedded class. OMAs are divided into regular and embedded, to distinguish mappings between object properties and mappings of object properties to an embedded class.

Based on the taxonomy presented in Fig. 6, we defined 7 mapping patterns. Now we exemplarily illustrate three patterns representative of the pattern library. These patterns are in accordance to the template defined in Section 5.1. We refer to the reader to Appendices Appendix A for a complete mapping pattern description.

The first pattern (shown below) models a recurring correspondence when mapping ontologies. The **Regular Class Mapping Pattern** occurs when a class in the ontology corresponds to a class in another ontology. The instances in the latter can be restricted to those that satisfy a given condition. For example, *mo:MusicArtist* corresponds to *dbo:MusicalArtist*. Also, *mo:Record* corresponds to *dbo:Album* with its property *dbo:releaseDate* restricted to the value after "2013-01-01".

## Regular Class Mapping Pattern:

- Name:
  - **Name**: Regular Class Mapping Pattern
  - **Alias**: Regular CMA Pattern
- Problem:
  - **Problem**: Let $C_\mathbf{T}$ and $C_\mathbf{S}$ be classes in $\mathbf{V_T}$ and $\mathbf{V_S}$, respectively. Instances of $C_\mathbf{S}$ is mapped to instances of $C_\mathbf{T}$. There could be a complete mapping (all instances of $C_\mathbf{S}$ are mapped) or a partial mapping (only some instances of $C_\mathbf{S}$ are mapped).
  - **Context**: This is probably the most common type of CMA (details in Section 5.2). The mapping is 1-1 and preserves the URI of the source resources.
- Solution:
  - **MAs:** $\psi$: $C_\mathbf{T} \equiv C_\mathbf{S}$ / $f$
    Example:
    $\psi_1$: *mo:MusicArtist* ≡ *dbo:MusicalArtist*
    $\psi_2$: *mo:Record* ≡ *dbo:Album* / *dbo:releaseDate*> "2013-01-01"^^xsd;date

  - **Mapping rules:** $C_\mathbf{T}(u) \leftarrow C_\mathbf{S}(u)$; $f(u)$
    Example:
    R1: *mo:MusicArtist*(*s*) ← *dbo:MusicalArtist*(*s*)
    R2: *mo:Record*(*s*) ← *dbo:Album*(*s*) ; *dbo:releaseDate*(*s*, *v*) > '2013-01-01'^^xsd;date

  - **R2R mappings:** Template T1 (see Table 9, page 20)
    Example (to $\psi_2$ only):

```
mp:ψ2 a r2r:ClassMapping ;
  r2r:prefixDefinitions "mo: <http://purl.org/ontology/mo/> .
                         dbo: <http://dbpedia.org/ontology/>" ;
  r2r:sourcePattern "?SUBJ a dbo:Album ; dbo:releaseDate ?t .
                     FILTER(?t > '2013-01-01'^^xsd;date)" ;
  r2r:targetPattern "?SUBJ a mo:Record".
```

The second mapping pattern solves the mapping between *mo:Label* and *myspo:recordLabel* presented in Section 3. We describe it here using the pattern template defined in Section 5.1.

## Embedded Class Mapping Pattern:

- Name:
  - **Name**: Embedded Class Mapping Pattern
  - **Alias**: Embedded CMA Pattern
- Problem:
  - **Problem**: Let $C_\mathbf{T}$ and $C_\mathbf{S}$ be classes in $\mathbf{V_T}$ and $\mathbf{V_S}$, respectively. Instances of an embedded class $C_\mathbf{S}[A_1,\ldots,A_n]$ in $\mathbf{O_S}$ is mapped to instances $C_\mathbf{T}$. There could be a complete mapping (all instances of $C_\mathbf{S}$ are mapped) or a partial mapping (only some instances of $C_\mathbf{S}$ are mapped).
  - **Context**: The mapping can be 1-1 or 1-n. The URI of the source resources ARE NOT preserved by the mapping. The target resource has not a *sameAs* relation with the source resource.
- Solution:
  - **MAs:** $\psi$: $C_\mathbf{T} \equiv C_\mathbf{S}[A_1,\ldots,A_n]$ / $f$
    Example:
    $\psi_3$: *mo:Label* ≡ *myspo:MusicArtist*[*myspo:recordLabel*]

  - **Mapping rules:** $C_\mathbf{T}(u) \leftarrow C_\mathbf{S}(s)$; generateUri[$\psi$](*s*,*u*); $f(s)$
    Example:
    R3: *mo:Label*(*u*) ← *myspo:MusicArtist*(*s*) ; generateUri[$\psi_3$](*s*,*u*)

- **R2R mappings:** Template T2 (see Table 9, page 20)
  Example:

```
mp:ψ3
a r2r:ClassMapping ;
  r2r:prefixDefinitions "mo: <···> . myspo:<···>" ;
  r2r:sourcePattern "?SUBJ a myspo:MusicalArtist" ;
  r2r:targetPattern "?u a mo:Label" ;
  r2r:transformation "?u = generateUri[ψ3](?SUBJ)" .
```

The following pattern (shown below) illustrates a common mapping needed when a property in one ontology is modeled as a path[20] in the other ontology. For example, *moa:birthPlace* corresponds to *dbp:birthPlace/dbp:Country*.

## Regular Object Property Mapping Pattern:

- Name:
  - **Name**: Regular Object Property Mapping Pattern
  - **Alias**: Regular OMA Pattern
- Problem:
  - **Problem**: Let $C_{\mathbf{T}}$ be a class in $\mathbf{V_T}$ and $C_{\mathbf{S}}$ be a class in $\mathbf{V_S}$. Let $P_{\mathbf{T}}$ be an object property whose domain is $C_{\mathbf{T}}$ (or a superclass of $C_{\mathbf{T}}$) and $P_{\mathbf{S}}$ be an object property whose domain is $C_{\mathbf{S}}$ (or a superclass of $C_{\mathbf{S}}$). Instances of the property $P_{\mathbf{S}}$ is mapped to instances of $P_{\mathbf{T}}$.There could be a complete mapping (all instances of $P_{\mathbf{S}}$ are mapped) or a partial mapping (only some instances of $P_{\mathbf{S}}$ are mapped). $P_{\mathbf{T}}$ and $P_{\mathbf{S}}$ can have the same or different names. $P_{\mathbf{S}}$ can be achieved through a path $\varphi$ from $C_{\mathbf{S}}$ to another class of $\mathbf{V_S}$.
  - **Context**: This is the most common type of MAs between object properties.
- Solution:
  - **MAs:** $\psi$: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/P_{\mathbf{S}}/f$
    Constraints:
    If the OMA is of the form $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/P_{\mathbf{S}}/f$, then must exists a CMA that matches the domain of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$, and a CMA that matches the range of $P_{\mathbf{T}}$ with the range of $P_{\mathbf{S}}$. If the OMA is of the form $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/f$, where $\varphi$ is a path from $C_{\mathbf{S}}$ to $C_{\mathbf{R}}$, then must exist a CMA that matches the domain of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$ and a CMA that matches the range of $P_{\mathbf{T}}$ with $C_{\mathbf{R}}$.
    Example:
    $\psi_4$: *mo:MusicArtist* / *moa:birthPlace* $\equiv$ *dbo:MusicalArtist* / $\varphi$, where
    $\varphi$= [*dbp:birthPlace/dbp:country*]

  - **Mapping rules:**
    $P_{\mathbf{T}}(u,v) \leftarrow C_{\mathbf{S}}(u)$ ; $f(u)$ ; $\varphi(u,s)$ ; $P_{\mathbf{S}}(s,v)$ ; $f(v)$
    Example:
    R4: *moa:birthPlace*$(s,v) \leftarrow$ *dbo:MusicalArtist*$(s)$; *dbp:birthPlace/dbp:country*$(s,v)$

  - **R2R mappings:** Template T3 (see Table 9, page 20)
    Example:

```
mp:ψ4 a r2r:PropertyMapping ;
  r2r:prefixDefinitions "mo: <http://purl.org/ontology/mo/> .
                         dbo: <http://dbpedia.org/ontology/>" ;
  r2r:mappingRef mp:ψ3 ;
  r2r:sourcePattern "?SUBJ dbp:birthPlace ?t . ?t dbp:country ?v" ;
  r2r:targetPattern "?SUBJ moa:birthPlace ?v".
```

[20] in particular, a path can have a single property.

Note that, the mapping patterns are broaden patterns. They can cover various specific type of mappings. For example, a mapping pattern between classes can include all instances of one class or only some instances (i.e., the mapping can include a filter or not). In similar way, a mapping can include (or not) paths and transformation functions. Thus, we cover all types of MAs shown in definitions 2, 3 and 4.

In the remain of this article, we show how mapping patterns can be instantiated and used to automatically generate R2R mappings.

## 6 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 the step 2, in order to load the target ontology.

In the current work, the MAs are manually specified. However, we use the mapping patterns and the *MAs_editor* (will be explained latter) in order 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 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. Algorithm 1 shows the procedure to automatically generate the statements of R2R mappings from MAs in $\mathcal{M}$.

**Algorithm 1** Generate the R2R mapping from MAs.

```
1: for each class C_T in O_T do
2:     G_R2RclassMapping(C_T)
3:     for each object property P_T whose domain is C_T do
4:         G_OMA_R2RpropMapping(P_T)
5:     end for
6:     for each datatype property P_T whose domain is C_T do
7:         G_DMA_R2RpropMapping(P_T)
8:     end for
9: end for
```

The algorithm 1 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* (shown in algorithm 2). 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* (shown in algorithm 4).

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* (shown in algorithm 3).

**Algorithm 2** G_R2RclassMapping().

**Input:** $C_{\mathbf{T}}$

1: **for each** CMA $\psi_C$ of $C_{\mathbf{T}}$ **do**
2: sQuery = NULL
3: prefixExp = getPrefixes($\psi_C$) ▷ get prefixes presents into $\psi_C$
4: **if** $\psi_C$ is of form $\psi_C$: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}/f$ **then** ▷ $\psi_C$ is a regular CMA mapping pattern
5: **if** $f \neq$ NULL **then**
6: sQuery = sQuery + FilterExp($\psi_C$)
7: **end if**
8: use template T1
9: **else** ▷ $\psi_C$ is an embedded CMA map. pattern: $\psi_C$:$C_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1, \ldots, A_n]/f$
10: **if** $f \neq$ NULL **then**
11: sQuery = sQuery + FilterExp($\psi_C$)
12: **end if**
13: use template T2
14: **end if**
15: **end for**

In algorithm 2, the R2R mapping is generated using the templates T1 or T2 (shown in Table 9) in accordance to the type of mapping pattern. Both templates use two variables that will be bound with values obtained from the CMA, and/or from the ontologies. These variable 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.

Figure 7 presents the R2R mapping to map *mo:Record* with *dbo:Album*. This mapping was generated by $\psi_2$ using the template T1. The ***r2r:prefixDefinitions*** clause (line 03) was obtained from $\psi_2$ in addition with **Music** ontology and **DBpedia** ontology. The ***r2r:sourcePattern*** clause (line 04) was obtained from the right hand side of the CMA $\psi_2$, being that **S:$C_{\mathbf{S}}$** = *dbo:Album* and sQuery = ‘‘; *dbo:releaseDate* ?t . FILTER(?t > ‘2013-01-01’). The ***r2r:targetPattern*** clause (line 05) was obtained from the left hand side of the CMA $\psi_2$, being that **T:$C_{\mathbf{T}}$** = *mo:Record*.

```
01. mp:ψ2
02. a r2r:ClassMapping;
03.   r2r:prefixDefinitions "mo: <http://purl.org/ontology/mo/> .
                             dbo: <http://dbpedia.org/ontology/>";
04.   r2r:sourcePattern "?SUBJ a dbo:Album ; dbo:releaseDate ?t .
                         FILTER(?t > '2013-01-01'^^xsd:date)";
05.   r2r:targetPattern "?SUBJ a mo:Record" .
```

Fig. 7: R2R mappings generated by $\psi_2$ using template T1.

**Algorithm 3** G_OMA_R2RpropMapping().

**Input:** $P_{\mathbf{T}}$

1: **for each** OMA $\psi_P$ of $P_{\mathbf{T}}$ **do**
2: sQuery = NULL
3: prefixExp = getPrefixes($\psi_P$) ▷ get prefixes presents into $\psi_P$
4: $\psi_C$ = foundCMA($\psi_P$) ▷ get the CMA of the domain of $P_{\mathbf{T}}$
5: **if** $\psi_P$ is of form $\psi_P$: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}//\varphi/P_{\mathbf{S}}/f$ **then** ▷ $\psi_P$ is a regular OMA map. pattern
6: **if** $\varphi \neq$ NULL **then**
7: sQuery = sQuery + Path($\varphi$) + "; S:$P_{\mathbf{S}}$ ?$P_{\mathbf{S}}$"
8: **else** sQuery = sQuery + "S:$P_{\mathbf{S}}$ ?$P_{\mathbf{S}}$"
9: **end if**
10: **if** $f \neq$ NULL **then**
11: sQuery = sQuery + FilterExp($\psi_P$)
12: **end if**
13: use template T3
14: **else** use template T5 ▷ $\psi_P$ is an embedded OMA map. pat.: $\psi_P$:$C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1, \ldots, A_n]$/NULL
15: **end if**
16: **end for**

In the algorithm 3, the R2R mapping is generated using the template T3 or T5 (shown in Table 9) in accordance to the type of mapping pattern. Both templates also use the variables prefixExp and sQuery. In addition, the algorithm also use the variable $\psi_C$, which keeps the CMA that is the domain of the object property being examined. Whether the OMA is a regular mapping pattern, then the source expression can have a path and/or a filter. When the OMA has a path $\varphi$, *Path*($\varphi$) is used to convert $\varphi$ to the R2R syntax and its result is concatenated to the value of sQuery. When the OMA has a filter $f$, *FilterExp* is used (as explained before).

Figure 8 presents the R2R mapping to map *moa:birthPlace* with *dbp:country*. This mapping was generated by $\psi_4$ using the template T3. The ***r2r:prefixDefinitions*** clause (line 03) was obtained from $\psi_4$ in addition with **Music** ontology and the **DBpedia** ontology. The ***r2r:mappingRef*** clause (line 04) was obtained from the value of $\psi_C$. The ***r2r:sourcePattern*** clause (line 05) was obtained from the right hand side of the OMA $\psi_4$, being that the expression "dbp:birthPlace ?d . ?d dbp:country ?country" is the output of *Path*($\varphi$). The ***r2r:targetPattern*** clause (line 06) was obtained from the left hand side of the OMA $\psi_4$, being that **T:$P_{\mathbf{T}}$** in the template is replaced by *moa:birthPlace* in the query.

```
01. mp: ψ4
02. a r2r:PropertyMapping;
03.   r2r:prefixDefinitions "moa: <···> . dbp: <···>" ;
04.   r2r:mappingRef mp:ψ1 ;
05.   r2r:sourcePattern "?SUBJ dbp:birthPlace ?birthPlace .
                         ?birthPlace dbp:country ?country";
06.   r2r:targetPattern "?SUBJ moa:birthPlace ?country" .
```

Fig. 8: Example of R2R mapping generated by $\psi_4$ using template T3.

**Algorithm 4** G_DMA_R2RpropMapping().

**Input:** $P_{\mathbf{T}}$

1: **for each** DMA $\psi_P$ of $P_{\mathbf{T}}$ **do**
2: sQuery = NULL
3: prefixExp = getPrefixes($\psi_P$) ▷ get prefixes presents into $\psi_P$
4: $\psi_C$ = foundCMA($\psi_P$) ▷ get the CMA of the domain of $P_{\mathbf{T}}$
5: **if** $\psi_P$ is of form $\psi_P$:$C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/P_{\mathbf{S}}/f/\mathcal{T}$ **then** ▷ $\psi_P$ is a regular DMA map. pat.
6: **if** $\varphi \neq$ NULL **then**
7: sQuery = sQuery + Path($\varphi$) + "; S:$P_{\mathbf{S}}$ ?$P_{\mathbf{S}}$"
8: **else** sQuery = sQuery + "S:$P_{\mathbf{S}}$ ?$P_{\mathbf{S}}$"
9: **end if**
10: **if** $f \neq$ NULL **then**
11: sQuery = sQuery + FilterExp($\psi_P$)
12: **end if**
13: **if** $\mathcal{T} \neq$ NULL **then**
14: FunctionExp = Function($\mathcal{T}$)
15: use template T6
16: **else** use template T3
17: **end if**
18: **else**
19: **if** $\psi_P$ is of form $\psi_P$:$C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1, \ldots, A_n]/A_i$ **then** ▷ $\psi_P$ is an embedded DMA
20: sQuery = sQuery + "S:$A_i$ ?$A_i$"
21: use template T4
22: **else** ▷ $\psi_P$ is a multi DMA map. pattern of form: $\psi_P$:$C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/\{P_{\mathbf{S}_1}, \ldots, P_{\mathbf{S}_n}\}/\mathcal{T}$
23: **if** $\varphi \neq$ NULL **then**
24: sQuery = sQuery + Path($\varphi$) + "; S:$P_{\mathbf{S}_1}$ ?$P_{\mathbf{S}_1}$;...;S:$P_{\mathbf{S}_n}$ ?$P_{\mathbf{S}_n}$"
25: **else** sQuery = sQuery + "S:$P_{\mathbf{S}_1}$ ?$P_{\mathbf{S}_1}$;...;S:$P_{\mathbf{S}_n}$ ?$P_{\mathbf{S}_n}$"
26: **end if**
27: **if** $\mathcal{T} \neq$ NULL **then**
28: FunctionExp = Function($\mathcal{T}$)
29: use template T8
30: **else** use template T7
31: **end if**
32: **end if**
33: **end if**
34: **end for**

In the algorithm 4, the R2R mapping is generated using the template T3, T4, T6, T7 or T8 (shown in Table 9), in accordance to the type of mapping pattern involved. All templates use the same variables than those used in algorithm 3, except templates T6 and T8 that also use the variable FunctionExp. FunctionExp keeps the expression used to form the ***r2r:transformation*** clause to transform the value of the source before it is load in the target. A DMA can have a path $\varphi$ and/or a filter $f$ and/or a transformation function $\mathcal{T}$. In each case, we need to use, respectively, *Path*($\varphi$), *FilterExp* and *Function*($\mathcal{T}$). The two first were already discussed. *Function*($\mathcal{T}$) is used to convert $\mathcal{T}$ to the R2R syntax and its result is used to form the *r2r:transform* clause.

Figure 9 presents the R2R mapping to map *moa:careerDuration* with the properties *dbp:startCareer* and *dbo:endCareer* generated by $\psi_7$ using the template T7. The ***r2r:mappingRef*** clause (line 04) was obtained from the value of $\psi_C$. The ***r2r:sourcePattern*** clause (line 05) was obtained from the right hand side of the DMA $\psi_7$. The ***r2r:targetPattern*** clause (line 06) was obtained from the left hand side of the DMA $\psi_7$, being that **T:$P_{\mathbf{T}}$** in the template is replaced by *moa:careerDuration*. The ***r2r:transformation*** clause (line 07) was obtained directly from the the template 7, being the property names were obtained from the right hand side of the DMA $\psi_7$.

```
01. mp: ψ7
02. a r2r:PropertyMapping;
03.   r2r:prefixDefinitions "moa: <···> . dbp: <···>" ;
04.   r2r:mappingRef mp:ψ1 ;
05.   r2r:sourcePattern "?SUBJ dbp:startCareer ?startCareer ;
                         dbp:endCareer ?endCareer" ;
06.   r2r:targetPattern "?SUBJ moa:careerDuration ?v" ;
07.   r2r:transformation "?v= concat(?startCareer, ?endCareer)" .
```

Fig. 9: Example of R2R mapping generated by $\psi_7$ using template T7.

Table 9: Templates to generate R2R mappings induced by MAs

| **T1** | **T2** |
|---|---|
| # Class Mappings<br># CMA $\psi$: $C_T \equiv C_S/f$<br>mp: $\psi_C$<br>**a r2r:ClassMapping** ;<br>**r2r:prefixDefinitions** "prefixExp";<br>**r2r:sourcePattern** "?SUBJ a **S:**$C_S$ sQuery" ;<br>**r2r:targetPattern** "?SUBJ a **T:**$C_T$" . | # Class Mappings<br># CMA $\psi$: $C_T \equiv C_S[A_1,\ldots,A_n]/f$<br>mp: $\psi_C$<br>**a r2r:ClassMapping** ;<br>**r2r:prefixDefinitions** "prefixExp";<br>**r2r:sourcePattern** "?SUBJ a **S:**$C_S$ sQuery" ;<br>**r2r:targetPattern** "?SUBJ a **T:**$C_T$" ;<br>**r2r:transformation** "?s=GenerateUri[$\psi_C$](?SUBJ)". |
| **T3** | **T4** |
| # Property Mappings<br># DMA/OMA $\psi$:$C_T/P_T \equiv C_S/\varphi/P_S/f$<br>mp: $\psi_P$<br>**a r2r:PropertyMapping** ;<br>**r2r:prefixDefinitions** "prefixExp";<br>**r2r:mappingRef** mp: $\psi_C$ ;<br>**r2r:sourcePattern** "?SUBJ sQuery" ;<br>**r2r:targetPattern** "?SUBJ **T:**$P_T$ ?$P_S$" . | # Property Mappings<br># DMA $\psi$: $C_T/P_T \equiv C_S[A_1 \ldots A_n]/A_i/f$<br>mp: $\psi_P$<br>**a r2r:PropertyMapping** ;<br>**r2r:prefixDefinitions** "prefixExp";<br>**r2r:mappingRef** mp: $\psi_C$ ;<br>**r2r:sourcePattern** "?SUBJ sQuery" ;<br>**r2r:targetPattern** "?s **T:**$P_T$ ?$A_i$" ;<br>**r2r:transformation** "?s=GenerateUri[$\psi_C$](?SUBJ)". |
| **T5** | **T6** |
| # Property Mappings<br># OMA $\psi$: $C_T/P_T \equiv C_S[A_1 \ldots A_n]$/**NULL**<br>mp: $\psi_P$<br>**a r2r:PropertyMapping** ;<br>**r2r:prefixDefinitions** "prefixExp";<br>**r2r:mappingRef** $\psi_C$ ;<br>**r2r:sourcePattern** "?SUBJ a **T:**$C_T$" ;<br>**r2r:targetPattern** "?SUBJ **T:**$P_T$ ?s" ;<br>**r2r:transformation** "?s=GenerateUri[$\psi_R$](?SUBJ)". | # Property Mappings<br># DMA $\psi$: $C_T/P_T \equiv C_S/\varphi/P_S/f/\mathcal{T}$<br>mp: $\psi_P$<br>**a r2r:PropertyMapping** ;<br>**r2r:prefixDefinitions** "prefixExp";<br>**r2r:mappingRef** mp: $\psi_C$ ;<br>**r2r:sourcePattern** "?SUBJ sQuery" ;<br>**r2r:targetPattern** "?SUBJ **T:**$P_T$ ?v" ;<br>**r2r:transformation** "?v = FunctionExp" . |
| **T7** | **T8** |
| # Property Mappings<br># DMA $\psi$: $C_T/P_T \equiv C_S/\varphi/\{P_{S_1},\ldots,P_{S_n}\}$<br>mp: $\psi_P$<br>**a r2r:PropertyMapping** ;<br>**r2r:prefixDefinitions** "prefixExp";<br>**r2r:mappingRef** mp: $\psi_C$ ;<br>**r2r:sourcePattern** "?SUBJ sQuery" ;<br>**r2r:targetPattern** "?SUBJ **T:**$P_T$ ?v" ;<br>**r2r:transformation** "?v=concat(?$P_{S_1}$,…,?$P_{S_n}$)". | # Property Mappings<br># DMA $\psi$: $C_T/P_T \equiv C_S/\varphi/\{P_{S_1},\ldots,P_{S_n}\}/\mathcal{T}$<br>mp: $\psi_P$<br>**a r2r:PropertyMapping** ;<br>**r2r:prefixDefinitions** "prefixExp";<br>**r2r:mappingRef** mp: $\psi_C$ ;<br>**r2r:sourcePattern** "?SUBJ sQuery" ;<br>**r2r:targetPattern** "?SUBJ **T:**$P_T$ ?v" ;<br>**r2r:transformation** "?w=concat(?$P_{S_1}$,…,?$P_{S_n}$)";<br>**r2r:transformation** "?v=FunctionExp". |

# 7 Implementation

This section presents RBA (*R2R By Assertions*), a tool based on mapping patterns to automatically generate R2R mappings. RBA has been developed using Maven and Java. We employed the open-source Apache Jena API[21] to process and manipulate the Semantic Web data. In addition, we use the R2R framework[22] to create the RDF triples using the R2R mappings generated by RBA.

Figure 10 depicts the main components of RBA. Briefly, the designer should define a set of MAs using RBA (*R2R By Assertions*) between the target and the source ontologies - Figure 10(a). These MAs are responsible for: (i) generate mapping rules between the target and the source ontologies; and (ii) generate R2R mappings between the target and the source ontologies. Then, the R2R mappings are used by the *R2R framework* to materialize source RDF triples into target format - Figure 10(b).

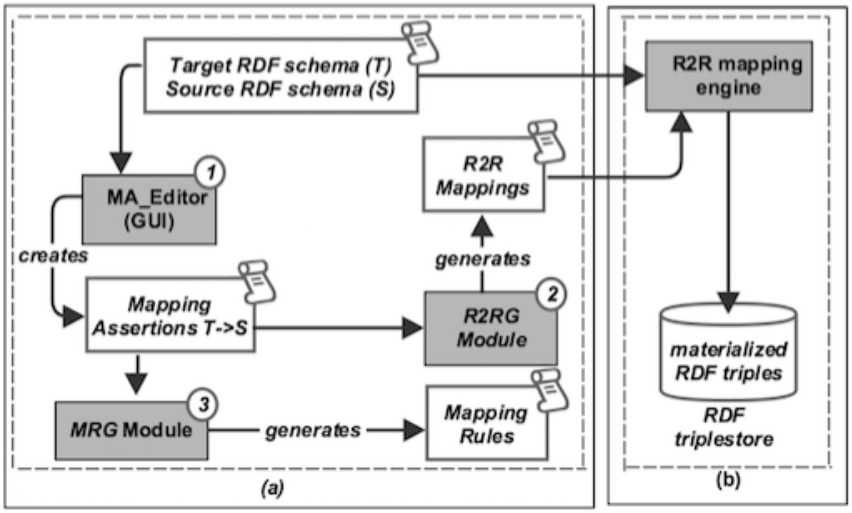


Fig. 10: Suggested Architecture.

The process of generating R2R mappings with RBA consists of three steps:

**STEP 1 (Mapping Assertion Specification):** Using the *MA_editor* of RBA, the designer loads the source and the target schemas and then he can draw MAs to specify the mapping between the target RDF schema and the source RDF schema.

**STEP 2 (R2R Mapping Generation):** The *R2RG module* automatically generates the R2R mappings required to load the target ontology in accordance to MA defined in Step 1. The R2R mappings are generated based on the templates that were defined in the mapping patterns, which the MAs are signed.

[21] https://jena.apache.org/

[22] Available in: http://wifo5-03.informatik.uni-mannheim.de/bizer/r2r/ .

**STEP 3 (MRG Module):** Optionally, the designer also can generate the mapping rules, for which the MAs are associated. This is done through the *MRG module*.

## 8 Related Work

Regarding ontology mapping, using design patterns, there are two related aspects: *ontology matching/alignment* and *design patterns in the ontologies*.

Ontology matching/alignment concerns on developing formalism to represent correspondences between ontology entities (e.g., classes and properties). Its purpose is on how one entity can be transformed to conceptually similar other entity. There are a lot of works in ontology matching (see [Shvaiko and Euzenat, 2013] for a survey). We can cite [Aumueller et al., 2005, Seligman et al., 2010, Leme et al., 2010] as example. Our approach differs from the ontology matching since that ontology mapping focus on capturing information on which entity can be transformed into another one and how this can be done. This implies that the mapping specifies how instances of an ontology can be transformed to instances of the other ontology. More general information handled in ontology matching, such as *match two classes whose set of instances overlapping*, is not important in the context of ontology mapping, since that this information is not enough to translate data from one vocabulary to another.

The most closely related work is LDIF [Schultz et al., 2011], which focus on discovering, combining and publishing mappings from different ontologies. Specifically, it provides a language (the R2R) for defining the relationship across the ontologies. R2R is a very expressive language, but not intuitive for the user and LDIF does not address the problem of how the mappings are defined. Besides, the definition of the R2R mappings is error-prone and time consuming (mainly when complex mappings are required). In this paper, we propose to use some types of mapping rules, those defined in the mapping patterns, to automatically generate R2R mappings. Thus, our approach can be view as a complement to the LDIF work.

Design patterns in the ontology was firstly used for ontology engineering (c.f. [Blomqvist and Sandkuhl, 2005]). There are a few mentions of design patterns in the ontology matching area (e.g., [Sequeda et al., 2012, Ritze et al., 2009, Scharffe, 2009, Sváb-Zamazal et al., 2009, Scharffe et al., 2014]) and, to the best of our knowledge, none in ontology mapping field. In [Sequeda et al., 2012], it is proposed patterns to reuse common R2RML mappings (matchings between RDBs and RDF). The structure of their patterns is very similar do ours, being that the matching problem addressed by the pattern is expressed by a question. [Ritze et al., 2009, Scharffe, 2009] deal with complex correspondences, which are captured as correspondence patterns. Correspondence patterns are templates that are used to help to find more specific correspondences than simply relating one entity to another. [Sváb-Zamazal et al., 2009] proposes an ontology transformation service based on patterns. In that work, parts of two ontologies are aligned and transformed into a new one using correspondence/transformation patterns. This differs of our approach since we do not want to align two ontologies, instead of, we want populate instances from source ontologies to a target ontology. Our mapping patterns are used to determine how the source instances are transformed into target instances.

[Scharffe et al., 2014] introduces ontology alignment patterns as a means to formalize solutions to ontology mismatches. Again, this scenario (the same that in [Sváb-Zamazal et al., 2009]) differs from our scenario. In that work, the relation of correspondence between entities is, in general, more strong that ours because the relation of correspondence needs to be valid in both ways. In our approach, the mapping is only in one way (from the sources to the target). In addition, there are some patterns in the [Scharffe et al., 2014] that are not necessary in our scenario, since that they only indicate types of relationship that do not can be used to obtain instances (e.g., patterns indicating that two classes are disjoint to each other). We inspired our mapping patterns in some alignment patterns presented in [Scharffe et al., 2014] and add new ones (e.g., mapping patterns to mapping embedded classes with filters). Our approach also differs to [Scharffe et al., 2014] in two aspects: 1) we use a formal notation when show the solutions to the problems addressed by the patterns; and 2) we present, when it is the case, the constraints to use the patterns.

## 9 Conclusions

This paper presents a proposal to automatically generate R2R mappings using mapping patterns. We briefly introduced Mapping Assertions (MAs) as a high-level language to specify mappings between RDF vocabularies. We emphasize that, in our approach, the MAs can specify basic and complex mappings with semantics. Using MAs, which are part of mapping patterns, we shown how R2R mappings can be automatically generated to populate a target ontology based on sources ontologies. Also, in this paper, we briefly show RBA, a tool for helping the designer in the process of definition of mappings. Actually, RBA enables to define only some types of MAs. Ongoing work includes to finish the implementation of RBA by covering all types of MAs included in the mapping patterns proposed in this article. As future work, we will test the usability of our proposal in real world scenarios.

## ACKNOWLEDGEMENTS

This work has been supported by national funds through FCT - Fundação para a Ciência e a Tecnologia, under the project PEst-OE/EEI/LA0021/2013, DataStorm Research Line of Excellency funding (EXCL/EEI-ESS/0257/2012) and the grant SFRH/BPD/76024/2011.

## References

Aumueller et al., 2005. Aumueller, D., Do, H.-H., Massmann, S., and Rahm, E. (2005). Schema and ontology matching with COMA++. In *SIMOD'05*, pages 906–908, New York, NY, USA. ACM.

Bizer and Schultz, 2010. Bizer, C. and Schultz, A. (2010). The r2r framework: Publishing and discovering mappings on the web. In Hartig, O., Harth, A., and Sequeda, J., editors, *COLD*, volume 665 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Blomqvist and Sandkuhl, 2005. Blomqvist, E. and Sandkuhl, K. (2005). Patterns in ontology engineering: Classification of ontology patterns. In *ICEIS (3)*, pages 413–416. Citeseer.

Klein, 2001. Klein, M. (2001). Combining and relating ontologies: an analysis of problems and solutions. In *IJCAI-2001 Workshop on Ontologies and Information Sharing*, pages 53–62, Seattle, WA.

Leme et al., 2010. Leme, L. A. P. P., Casanova, M. A., Breitman, K. K., and Furtado, A. L. (2010). Owl schema matching. *Journal of the Brazilian Computer Society*, 16(1):21–34.

Lutz, 2002. Lutz, C. (2002). Description logics with concrete domains—a survey. In *Advances in Modal Logic 2002 (AiML 2002)*, Toulouse, France.

Pequeno et al., 2015. Pequeno, V., Vidal, V., Vinuto, T., and Galhardas, H. (2015). Automatic generation of r2r mappings from correspondence assertions. In *SBBD (poster)*.

Ritze et al., 2009. Ritze, D., Meilicke, C., Šváb-Zamazal, O., and Stuckenschmidt, H. (2009). A pattern-based ontology matching approach for detecting complex correspondences. In *Proceedings of the 4th International Conference on Ontology Matching - Volume 551*, OM'09, pages 25–36, Aachen, Germany, Germany. CEUR-WS.org.

Scharffe, 2009. Scharffe, F. (2009). *Correspondence Patterns Representation*. PhD thesis, University of Innsbruck.

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

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

Seligman et al., 2010. Seligman, L., Mork, P., Halevy, A., Smith, K., Carey, M. J., Chen, K., Wolf, C., Madhavan, J., Kannan, A., and Burdick, D. (2010). Openii: an open source information integration toolkit. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD'10, pages 1057–1060, New York, NY, USA. ACM.

Sequeda et al., 2012. Sequeda, J., Priyatna, F., and Villazón-Terrazas, B. (2012). Relational database to rdf mapping patterns. In *WOP - Proceedings of the 3rd Workshop on Ontology Patterns*.

Shvaiko and Euzenat, 2013. Shvaiko, P. and Euzenat, J. (2013). Ontology matching: State of the art and future challenges. *Knowledge and Data Engineering, IEEE Transactions on*, 25(1):158–176.

Sváb-Zamazal et al., 2009. Sváb-Zamazal, O., Svátek, V., and Scharffe, F. (2009). Pattern-based ontology transformation service. In *Proc. 1st IK3C international conference on knowledge engineering and ontology development (KEOD)*, Proc. 1st IK3C international conference on knowledge engineering and ontology development (KEOD), pages 210–223, Funchal, Portugal. No commercial editor.

Vidal et al., 2013. Vidal, V. M. P., Casanova, M. A., and Cardoso, D. S. (2013). Incremental maintenance of RDF views of relational data. In *Proc. The 12th International Conference on Ontologies, DataBases, and Applications of Semantics*, Graz, Austria.

Vidal et al., 2001. Vidal, V. M. P., Lóscio, B. F., and Salgado, A. C. (2001). Using correspondence assertions for specifying the semantics of XML-based mediators. In *Workshop on Information Integration on the Web*, pages 3–11.

Šváb, 2007. Šváb, O. (2007). Exploiting patterns in ontology mapping. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 956–960, Berlin, Heidelberg. Springer-Verlag.

# *Appendix A - Mapping Patterns*

## A.1 Regular Class Mapping Pattern

- Name:
  - **Name**: Regular Class Mapping Pattern
  - **Alias**: Regular CMA Pattern
- Problem:
  - **Problem**: Let $C_{\mathbf{T}}$ and $C_{\mathbf{S}}$ be classes in $\mathbf{V_T}$ and $\mathbf{V_S}$, respectively. Instances of $C_{\mathbf{S}}$ is mapped to instances of $C_{\mathbf{T}}$. There could be a complete mapping (all instances of $C_{\mathbf{S}}$ are mapped) or a partial mapping (only some instances of $C_{\mathbf{S}}$ are mapped).
  - **Context**: This is probably the most common type of CMA. The mapping is 1-1 and preserves the URI of the source resources.
- Solution:
  - **MAs:** $\psi$: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}$ / $f$
    Example:
    $\psi_1$: *mo:MusicArtist* $\equiv$ *dbo:MusicalArtist*
    $\psi_2$: *mo:Record* $\equiv$ *dbo:Album* / *dbo:releaseDate*$>$ "2013-01-01"

  - **Mapping rules:** $C_{\mathbf{T}}(u) \leftarrow C_{\mathbf{S}}(u); f(u)$
    Example:
    R1: *mo:MusicArtist*($s$)$\leftarrow$*dbo:MusicalArtist*($s$)
    R2: *mo:Record*($s$) $\leftarrow$ *dbo:Album*($s$) ; *dbo:releaseDate*($s$, $v$) $>$ '2013-01-01'

- **R2R mappings:** Template T1 (see Table 9, page 20)
  Example:
  see mp:$\psi_1$ and mp:$\psi_2$ in Appendice Appendix B.

## A.2 Embedded Class Mapping Pattern

– Name:
  - **Name**: Embedded Class Mapping Pattern
  - **Alias**: Embedded CMA Pattern
– Problem:
  - **Problem**: Let $C_\mathbf{T}$ and $C_\mathbf{S}$ be classes in $\mathbf{V_T}$ and $\mathbf{V_S}$, respectively. Instances of a embedded class $C_\mathbf{S}[A_1,\ldots,A_n]$ in $\mathbf{O_S}$ is mapped to instances $C_\mathbf{T}$. There could be a complete mapping (all instances of $C_\mathbf{S}$ are mapped) or a partial mapping (only some instances of $C_\mathbf{S}$ are mapped).
  - **Context**: The mapping can be 1-1 or 1-n. The URI of the source resources ARE NOT preserved by the mapping. The target resource has not a *sameAs* relation with the source resource.
– Solution:
  - **MAs:** $\psi$: $C_\mathbf{T} \equiv C_\mathbf{S}[A_1,\ldots,A_n]$ / $f$
    Example:
    $\psi_3$: *mo:Label* ≡ *myspo:MusicArtist*[*myspo:recordLabel*]

  - **Mapping rules:** $C_\mathbf{T}(u) \leftarrow C_\mathbf{S}(s)$; generateUri[$\psi$]($s$,$u$); $f(s)$
    Example:
    R3: *mo:Label*($u$)←*myspo:MusicArtist*($s$) ; generateUri[$\psi_3$]($s$,$u$)

  - **R2R mappings:** Template T2 (see Table 9, page 20)
    Example:
    see mp:$\psi_3$ in Appendice Appendix B.

## A.3 Regular Datatype Property Mapping Pattern

– Name:
  - **Name**: Regular Datatype Property Mapping Pattern
  - **Alias**: Regular DMA Pattern
– Problem:
  - **Problem**: Let $C_\mathbf{T}$ be a class in $\mathbf{V_T}$ and $C_\mathbf{S}$ be a class in $\mathbf{V_S}$. Let $P_\mathbf{T}$ be a datatype property whose domain is $C_\mathbf{T}$ (or a superclass of $C_\mathbf{T}$) and $P_\mathbf{S}$ be a datatype property whose domain is $C_\mathbf{S}$ (or a superclass of $C_\mathbf{S}$). Instances of the property $P_\mathbf{S}$ is mapped to instances of $P_\mathbf{T}$.There could be a complete mapping (all instances of $P_\mathbf{S}$ are mapped) or a partial mapping (only some instances of $P_\mathbf{S}$ are mapped). $P_\mathbf{T}$ and $P_\mathbf{S}$ can have the same or different names. $P_\mathbf{S}$ can be achieved through a path $\varphi$ from $C_\mathbf{S}$ to another class of $\mathbf{V_S}$. Also, it can be necessary to use some transformation function to deal with the different encoding of the values in the source and the target ontologies, conversion between different currencies, unit of measures, data type formats, etc..
  - **Context**: This is the most common type of MAs between datatype properties.
– Solution:
  - **MAs:** $\psi$: $C_\mathbf{T}/P_\mathbf{T} \equiv C_\mathbf{S}/\varphi/P_\mathbf{S}/f/T$
    Constraints:
    It must exists a CMA that matches the domain of $P_\mathbf{T}$ with the domain of $P_\mathbf{S}$.
    Example:
    $\psi_5$: *mo:Record /dc:title* ≡*dbo:Album /dbo:originalTitle*

  - **Mapping rules:**
    $P_\mathbf{T}(s,t) \leftarrow C_\mathbf{S}(s)$ ; $f(s)$ ; $\varphi$]($s$,$o$) ; $P_\mathbf{S}(o,v)$; $f(v)$; $T(v,t)$
    where: filter $f(s)$ is optional and only occurs in the mapping rule when the CMA that match the domain of $P_\mathbf{T}$ include a filter.
    Example:
    R5: *dc:title*($s$,$v$)←*dbo:Album*($s$) ; *dbo:originalTitle*($s$,$v$)

  - **R2R mappings:** Templates T3 or T6
    Example:
    see mp:$\psi_5$ in Appendice Appendix B.

## A.4 Multiple Datatype Property Mapping Pattern

- Name:
  - **Name**: Multiple Datatype Property Mapping Pattern
  - **Alias**: Multiple DMA Pattern
- Problem:
  - **Problem**: Let $C_{\mathbf{T}}$ be a class in $\mathbf{V_T}$ and $C_{\mathbf{S}}$ be a class in $\mathbf{V_S}$. Let $P_{\mathbf{T}}$ be a datatype property whose domain is $C_{\mathbf{T}}$ (or a superclass of $C_{\mathbf{T}}$) and $P_{\mathbf{S}_1}, \ldots, P_{\mathbf{S}_n}$ be datatype properties whose domain is $C_{\mathbf{S}}$ (or a superclass of $C_{\mathbf{S}}$). Instances of the properties $P_{\mathbf{S}_1}, \ldots, P_{\mathbf{S}_n}$ are mapped to instances of $P_{\mathbf{T}}$. $P_{\mathbf{S}}$ can be achieved through a path $\varphi$ from $C_{\mathbf{S}}$ to another class of $\mathbf{V_S}$. Also, it can be necessary to use some transformation function to deal with the different encoding of the values in the source and the target ontologies, conversion between different currencies, unit of measures, data type formats, etc..
  - **Context**: This is a common type of MAs between datatype properties and occurs when datatype properties correspond on n:1 basis, instead of 1:1.
- Solution:
  - **MAs:** $\psi$: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/\{P_{\mathbf{S}_1}, \ldots, P_{\mathbf{S}_n}\}/T$
    Constraints:
    It must exists a CMA that matches the domain of $P_{\mathbf{T}}$ with the domain of $P_{\mathbf{S}_1}, \ldots, P_{\mathbf{S}_n}$.
    Example:
    $\psi_7$: *mo:MusicArtist* / *moa:careerDuration* ≡ *dbo:MusicalArtist* /
    / {*dbp:startCareer*, *dbp:endCareer*}

  - **Mapping rules:**
    $P_{\mathbf{T}}(s,t) \leftarrow C_{\mathbf{S}}(s)$ ; $f(s)$ ; $\varphi](s,o)$ ; $P_{\mathbf{S}_1}(o, v_1)$;. . . ; $P_{\mathbf{S}_n}(o, v_n)$ ; concat($v_1, \ldots, v_n$,$v$); $T(v,t)$
    where: filter $f(s)$ is optional and only occurs in the mapping rule when the CMA that match the domain of $P_{\mathbf{T}}$ include a filter $f$.
    Example:
    R7: *moa:careerDuration*($s$,$v$)←*dbo:MusicalArtist*($s$); *dbp:startCareer*($s$,$v_1$);
    *dbp:endCareer*($s$,$v_2$); cocat($v_1$, $v_2$, $v$)

  - **R2R mappings:** Templates T7 or T8
    Example:
    see mp:$\psi_7$ in Appendice Appendix B.

## A.5 Embedded Datatype Property Mapping Pattern

- Name:
  - **Name**: Embedded Datatype Property Mapping Pattern
  - **Alias**: Embedded DMA Pattern
- Problem:
  - **Problem**: Let $C_{\mathbf{T}}$ be a class in $\mathbf{V_T}$ and $C_{\mathbf{S}}$ be a class in $\mathbf{V_S}$. Let $P_{\mathbf{T}}$ be a datatype property whose domain is $C_{\mathbf{T}}$ (or a superclass of $C_{\mathbf{T}}$) and $A_{\mathbf{S}_1}, \ldots, A_{\mathbf{S}_n}$ be datatype properties whose domain is $C_{\mathbf{S}}$ (or a superclass of $C_{\mathbf{S}}$). Let also, $C_{\mathbf{S}}[A_1, \ldots, A_n]$ be an embedded class in $\mathbf{O_S}$. Instances of the property $A_{\mathbf{S}_i}$, $1 \leq i \leq n$, is mapped to instances of $P_{\mathbf{T}}$.
  - **Context**: This is not a common type of MAs between datatype properties.
- Solution:
  - **CAs:** $\psi$: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1, \ldots, A_n]$ / $A_i$ ($1 \leq$ i $\leq$ n)
    Constraints:
    It must exists a CMA that matches the domain of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}[A_1, \ldots, A_n]$.
    Example:
    $\psi_6$: *mo:Label* / *moa:labelName* ≡ *myspo:MusicArtist*[*myspo:recordLabel*] /
    *myspo:recordLabel*

  - **Mapping rules:** $P_{\mathbf{T}}(u,v) \leftarrow C_{\mathbf{S}}(s)$; $f(s)$ ; $P_{\mathbf{S}}(s,v)$ ; generateUri[$\psi$]($s$,$u$)
    where: filter $f(s)$ is optional and only occurs in the mapping rule when the Class Correspondence Assertion (CCA) that match the domain of $P_{\mathbf{T}}$ include a filter.
    Example:
    R6: *moa:labelName*($u$,$w$)←*myspo:MusicArtist*($s$); *myspo:recordLabel*($s$,$w$);
    generateUri[$\psi_3$]($s$,$u$)

- **R2R mappings:** Template T4
  Example:
  see mp:$\psi_6$ in Appendice Appendix B.

## A.6 Regular Object Property Mapping Pattern

- Name:
  - **Name**: Regular Object Property Mapping Pattern
  - **Alias**: Regular OMA Pattern
- Problem:
  - **Problem**: Let $C_{\mathbf{T}}$ be a class in $\mathbf{V_T}$ and $C_{\mathbf{S}}$ be a class in $\mathbf{V_S}$. Let $P_{\mathbf{T}}$ be an object property whose domain is $C_{\mathbf{T}}$ (or a superclass of $C_{\mathbf{T}}$) and $P_{\mathbf{S}}$ be an object property whose domain is $C_{\mathbf{S}}$ (or a superclass of $C_{\mathbf{S}}$). Instances of the property $P_{\mathbf{S}}$ is mapped to instances of $P_{\mathbf{T}}$.There could be a complete mapping (all instances of $P_{\mathbf{S}}$ are mapped) or a partial mapping (only some instances of $P_{\mathbf{S}}$ are mapped). $P_{\mathbf{T}}$ and $P_{\mathbf{S}}$ can have the same or different names. $P_{\mathbf{S}}$ can be achieved through a path $\varphi$ from $C_{\mathbf{S}}$ to another class of $\mathbf{V_S}$.
  - **Context**: This is the most common type of MAs between object properties.
- Solution:
  - **MAs:** $\psi$: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/P_{\mathbf{S}}/f$
    Constraints:
    If the OMA is of the form $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/P_{\mathbf{S}}/f$, then must exists a CMA that matches the domain of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$, and a CMA that matches the range of $P_{\mathbf{T}}$ with the range of $P_{\mathbf{S}}$. If the OMA is of the form $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/f$, where $\varphi$ is a path from $C_{\mathbf{S}}$ to $C_{\mathbf{R}}$, then must exists a CMA that matches the domain of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$ and a CMA that matches the range of $P_{\mathbf{T}}$ with $C_{\mathbf{R}}$.
    Example:
    $\psi_4$: *mo:MusicArtist* / *moa:birthPlace* $\equiv$ *dbo:MusicalArtist* / $\varphi$, where
    $\varphi$= [*dbp:birthPlace/dbp:country*]

  - **Mapping rules:**
    $P_{\mathbf{T}}(u,v) \leftarrow C_{\mathbf{S}}(u)$ ; $f(u)$ ; $\varphi(u,s)$ ; $P_{\mathbf{S}}(s,v)$ ; $f(v)$
    Example:
    R4: *moa:birthPlace*$(s,v) \leftarrow$ *dbo:MusicalArtist*$(s)$; *dbp:birthPlace/dbp:country*$(s,v)$

  - **R2R mappings:** Template T3 (see Table 9, page 20)
    Example:
    see mp:$\psi_4$ in Appendice Appendix B.

## A.7 Embedded Object Property Mapping Pattern

- Name:
  - **Name**: Embedded Object Property Mapping Pattern
  - **Alias**: Embedded OMA Pattern
- Problem:
  - **Problem**: Let $C_{\mathbf{T}}$ be a class in $\mathbf{V_T}$ and $C_{\mathbf{S}}$ be a class in $\mathbf{V_S}$. Let $P_{\mathbf{T}}$ be a datatype property whose domain is $C_{\mathbf{T}}$ (or a superclass of $C_{\mathbf{T}}$). Let also, $C_{\mathbf{S}}[A_1,\ldots,A_n]$ be an embedded class in $\mathbf{O_S}$. Instances of the embedded class $C_{\mathbf{S}}[A_1,\ldots,A_n]$ are mapped to instances of $P_{\mathbf{T}}$.
  - **Context**: This is not a common type of MAs between object properties.
- Solution:
  - **MAs:** $\psi$: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1,\ldots,A_n]$ / NULL
    Constraints:
    It must exist a CMA that matches the domain of $P_{\mathbf{T}}$ with $C_{\mathbf{S}}$, and a CMA that matches the range of $P_{\mathbf{T}}$ also with $C_{\mathbf{S}}$.
    Example:
    $\psi_8$: *mo:MusicArtist* / *moa:hasLabel* $\equiv$ *myspo:MusicArtist*[*myspo:recordLabel*] / NULL
  - **Mapping rules:** $P_{\mathbf{T}}(s,u) \leftarrow C_{\mathbf{S}}(s)$; $f(s)$ ; generateUri[$\psi$]$(s,u)$
    where: filter $f(s)$ is optional and only occurs in the mapping rule when the CMA that match the domain of $P_{\mathbf{T}}$ include a filter.
    Example:
    R8: *moa:hasLabel*$(s,o) \leftarrow$ *myspo:MusicArtist*$(s)$; generateUri[$\psi_8$]$(s,o)$

- **R2R mappings:** Templates T5 (see Table 9, page 20)
  Example:
  see mp:$\psi_8$ in Appendice Appendix B.

# Appendix B - R2R Mappings

```
@prefix r2r: <http://www4.wiwiss.fu-berlin.de/bizer/r2r/> .
@prefix owl: <http://www.w3.org/2002/07/owl# > .
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix mo: <http://purl.org/ontology/mo/> .
@prefix myspo:<http://purl.org/ontology/myspace/> .
@prefix dbp:<http://dbpedia.org/property/> .
@prefix dc:<http://purl.org/dc/elements/1.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema# > .

# Made up Mapping publisher
@prefix mp: <http://www.example.org/> .

# Class Mappings

mp:ψ1
a r2r:ClassMapping ;
  r2r:prefixDefinitions "mo: <http://purl.org/ontology/mo/> .
                         dbo: <http://dbpedia.org/ontology/>" ;
  r2r:sourcePattern "?SUBJ a dbo:MusicalArtist" ;
  r2r:targetPattern "?SUBJ a mo:MusicArtist".

mp:ψ2
a r2r:ClassMapping ;
  r2r:prefixDefinitions "mo: <http://purl.org/ontology/mo/> .
                        dbo: <http://dbpedia.org/ontology/>" ;
  r2r:sourcePattern "?SUBJ a dbo:Album ; dbo:releaseDate ?t .
                    FILTER(?t > '2013-01-01'^^xsd:date)" ;
  r2r:targetPattern "?SUBJ a mo:Record".

mp:ψ3
a r2r:ClassMapping ;
  r2r:prefixDefinitions "mo: <···> . myspo:<···>" .
  r2r:sourcePattern "?SUBJ a myspo:MusicalArtist" ;
  r2r:targetPattern "?u a mo:Label" ;
  r2r:transformation "?u = generateUri[ψ3](?SUBJ)" .

# Property Mappings

mp:ψ4
a r2r:PropertyMapping ;
  r2r:prefixDefinitions "moa:<···> . dbp:<···>" ;
  r2r:mappingRef mp:ψ1 ;
  r2r:sourcePattern "?SUBJ dbp:birthPlace ?t . ?t dbp:country ?v" ;
  r2r:targetPattern "?SUBJ moa:birthPlace ?v".
```

```
mp:ψ5
a r2r:PropertyMapping ;
  r2r:prefixDefinitions "dbo:<···> . dc:<···>" ;
  r2r:mappingRef mp:ψ2 ;
  r2r:sourcePattern "?SUBJ dbo:originalTitle ?t" ;
  r2r:targetPattern "?SUBJ dc:title ?t".

mp:ψ6
a r2r:PropertyMapping ;
  r2r:prefixDefinitions "myspo:<···> . moa:<···>" ;
  r2r:mappingRef mp:ψ3 ;
  r2r:sourcePattern "?SUBJ myspo:recordLabel ?r" ;
  r2r:targetPattern "?u moa:labelName ?r" ;
  r2r:transformation "?u = generateUri[ψ3](?SUBJ)" .

mp:ψ7
a r2r:PropertyMapping ;
  r2r:prefixDefinitions "moa: <···> . dbp: <···>" ;
  r2r:sourcePattern "?SUBJ dbp:startCareer ?startCareer ; dbp:endCareer ?endCareer" ;
  r2r:mappingRef mp:ψ1 ;
  r2r:targetPattern "?SUBJ moa:careerDuration ?v" ;
  r2r:transformation "?v= concat(?startCareer,'-', ?endCareer)" .

mp:ψ8
a r2r:PropertyMapping ;
  r2r:prefixDefinitions "mo: <···> . moa: <···>" ;
  r2r:mappingRef mp:ψ3 ;
  r2r:sourcePattern "?SUBJ a mo:Label" ;
  r2r:targetPattern "?SUBJ moa:hasLabel ?s" ;
  r2r:transformation "?s = generateUri[ψ3](?SUBJ)" .
```