Ontology Alignment Design Patterns

François Scharffe[1], Ondřej Zamazal[2], Dieter Fensel[3]
[1] Institut National de Recherche
en Informatique et Automatique, Grenoble, France,
[2] University of Economics, Prague, Czech Republic
[3] University of Innsbruck, Austria

October 8, 2012

Abstract

Interoperability between heterogeneous ontological descriptions can be performed through ontology mediation techniques. At the heart of ontology mediation lies the alignment: a specification of correspondences between ontology entities. Ontology matching can bring some automation but are limited to finding simple correspondences. Design patterns have proven themselves useful to capture experience in design problems. In this article we introduce ontology alignment patterns as reusable templates of recurring correspondences. Based on a detailed analysis of frequent ontology mismatches we develop a library of common patterns. Ontology alignment patterns can be used to refine correspondences, either by the alignment designer, or via pattern detection algorithms. We distinguish three levels of abstraction for ontology alignment representation, going from executable transformation rules, to concrete correspondences between two ontologies, to ontology alignment patterns at the third level. We express patterns using an ontology alignment representation language making them ready to use in practical mediation tasks. We extract mismatches from vocabularies associated to datasets published as linked open data and we evaluate the ability of correspondence patterns to provide proper alignments for these mismatches. Finally we describe an application of ontology alignment patterns for an ontology transformation service.

Keywords: ontology alignment patterns, ontology mapping, ontology alignment, ontology mediation, design patterns, data integration

1 Introduction

1.1 Ontology alignment as a design problem

Ontology mediation consists in the set of processes enabling interoperability between data sources described under heterogeneous ontologies. Ontology mediation is crucial in order to ensure a smooth exchange of structured data and to provide interoperability between vocabularies annotating unstructured or semi-structured data. Depending on the data exchange scenario, different tasks are considered in ontology mediation: transformation of instance data from one ontology to another, transformation of queries addressed to one ontology into queries for another ontology, fusion of two knowledge bases described by two overlapping ontologies, or simply enhancing search results in a semantic annotation scenario. In some cases, ontology mediation is needed to perform operations on the ontology structure: merging two ontologies or relating two versions of an evolving ontology. In order to be performed, each task needs a specification of the correspondences between the overlapping parts in each ontology. Ontology alignment is the part of ontology mediation that focuses on building this specification.

In this article, we consider ontology alignment as a design problem, where each correspondence between pairs or sets of ontological entities needs to be constructed. We propose to use design patterns as helpers to model correspondences. Each correspondence pattern will represent a generic solution to a given alignment problem. In order to achieve this goal, it is necessary to identify what the problems are. We do so by compiling ontology mismatches found in the literature. Then we propose a model to represent these correspondence patterns so they can be used in order to assist ontology alignment design. Before, in the remaining of this section we will give a concrete example where correspondence patterns can help. We also give background information on the relationship between ontology alignment and database schema mapping.

1.2 Motivating example

Let us consider as an example the mediation between FOAF[1] and vCard[2] ontologies, both describing persons. In order to exchange data the ontologies first need to be matched. The alignment resulting from this phase is then used to transform instances from vCard to FOAF, as illustrated on Figure 1.

In the scenario depicted with Figure 1, the alignment is grounded into a data mediator (Le Phuoc et al, 2009) transforming data from one ontology to the other.

[1]Friend Of A Friend: `http://xmlns.com/foaf/spec/`
[2]vCard: `http://www.w3.org/TR/vcard-rdf`

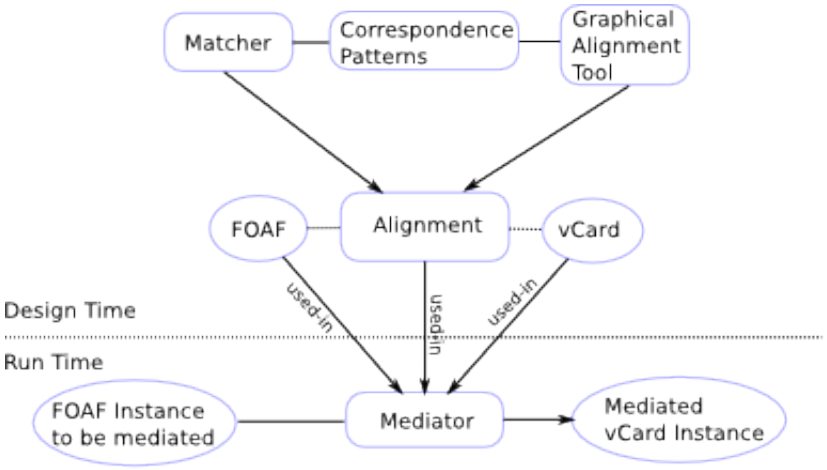

Figure 1: Data mediator between FOAF and VCard

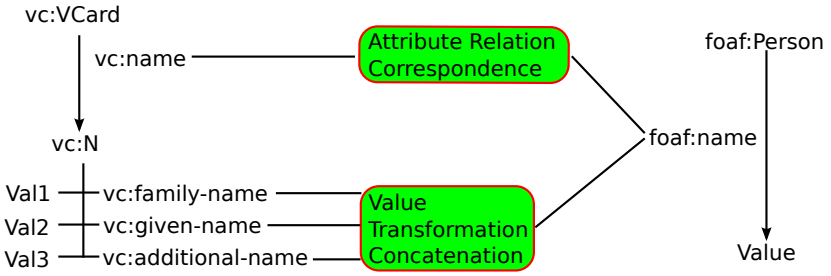

Figure 2: Correspondence between FOAF and vCard names

Grounding an alignment consists in transforming it into the executable language of a mediator.

As illustrated on Figure 2, in FOAF, a person's name is modeled as a string attached to the property `foaf:name`. A name in vCard is modeled as a relation `vcard:name` pointing to an object represented by the class `vCard:N`. This class itself has three properties `vCard:given-name`, `vCard:family-name`, and `vCard:additional-name` representing the different parts of a person's name. When aligning FOAF and vCard, the correspondence between names should capture the equivalence between these heterogeneous conceptualizations: the `foaf:name` property should be related to the `vCard:name` relation and to the concatenation of the three vCard name properties.

State of the art matching algorithms are not able to discover such a complex correspondence. Its manual design requires expertise on the two aligned ontologies, as well as expertise on the system used to design the alignment. This task can be long and tedious if the ontologies are large, but once designed an alignment

can be reused in various mediation scenarios, or to ease knowledge management processes (Oztemel and Arslankaya, 2012).

A library of ontology alignment patterns will facilitate ontology alignment design by providing templates modelling complex correspondences such as the one in the example given above. Used together with pattern matching techniques, patterns will also support the construction of better matching algorithms, as presented in Section 7.

1.3 Ontology alignment and database schema mapping

A large amount of work was realized in the field of relational schema mapping. Matching algorithms (Rahm and Bernstein, 2001), graphical interfaces (Chiticariu et al, 2007) and studies on representing mismatches and their solutions (Batini et al, 1986) were developed. Ontology alignment and relational schema mapping are indeed close fields, and the former can thus benefit from the results of the latter. Both communities have their own vocabularies for expressing corresponding concepts. Query translation in the database community is usually called query rewriting in the semantic web community, similarly, data exchange is called instance transformation, and record linkage is called instance matching (**?**; **?**).

An ontology in the semantic web context can be used in two ways that will influence the alignment process: in the first case, the ontology terms are used to annotate documents. This is a common usage in the biomedical field for example. In that case ontologies are mainly a list of terms with little structure; simple correspondences are sufficient to align the ontologies (see (Ghazvinian et al, 2009; Groza et al, 2011)). In the second case, the ontology is used to model data stored in a triple store. In a mediation scenario, information exchange or retrieval from two aligned ontologies will require more precise correspondences in the case of data oriented ontologies. This case will thus benefit more from database schema mapping techniques. Many alignment patterns studied in this article have also been studied under a different form in the data integration area (see for example (Ullman, 1997)).

The work presented in this paper is performed in the framework of the semantic web. Ontologies are thus RDFS schemas or OWL ontologies serialized in RDF. Data instances are represented in RDF according to these ontologies.[3] However, alignment patterns are generic enough to be adapted for other structures such as XML schemas. While usually one relational schema corresponds to one database, ontologies on the semantic web are meant to be reused (Gruber et al, 1995). Many datasets will thus be described according to the same ontology, resulting in dissociated schema and data. More important for the work presented in this article,

[3]These languages are W3C standards described in `http://www.w3.org/2001/sw/`

ontologies present structures such as class and property subsumption that do not appear in relational schemas. Finally, an approach to lift relational databases to the semantic web consists in exposing relational data as RDF, using a mapping to an ontology (Bizer, 2003).

We provide in this article foundations for pattern-based ontology alignment. In Section 2 we position our work in the fields of ontology alignment and design patterns. We give in Section 3 the representation formalism for correspondence patterns, based on a pattern template and a language for ontology alignment representation. We then present in Section 4 a library of patterns we have developed based on an empirical analysis of potential mismatches appearing when aligning ontologies. In Section 5 we use patterns in the library to solve ontology mismatches. In Section 6 we perform the alignment of ontologies modelling linked-data and show that correspondence patterns are useful to design precise alignments. In Section 7 we present a system using alignments patterns for transforming ontologies. Finally, we give conclusions to this work in Section 8.

2 Related work

A large amount of effort is currently spent on the ontology alignment problem, which has started with the integration of relational database schemas (Batini et al, 1986; Rahm and Bernstein, 2001; Doan and Halevy, 2005) and continues in ontology alignment (Kalfoglou and Schorlemmer, 2003; Shvaiko and Euzenat, 2005). Two areas can be distinguished: research on matching algorithms automatizing the similarity finding task, and research on efficient interfaces assisting the alignment process. Matching algorithms are efficient to finding simple correspondences, using terminological, structural and instance based techniques (Euzenat et al, 2007). Results of matching algorithms need to be refined, however, in order to provide alignments usable in practical mediation tasks (Shvaiko and Euzenat, 2008). Graphical user interfaces assisting the user to construct correspondences recently gained attention with the development of specific user interaction methodologies (Noy and Musen, 2000; McGuinness et al, 2000; Mocan et al, 2006; Falconer and Storey, 2007). As a large number of ontologies need to be aligned, community-based ontology alignment was proposed as a solution to facilitate the task (Zhdanova and Shvaiko, 2006; Noy et al, 2008), however, only simple one-to-one alignments are possible in these approaches. The work presented can be used either to perform more efficient pattern-based matching, or to improve graphical user interfaces by proposing patterns to the user.

Design patterns were first mentioned in architecture by Alexander in 1977 (Alexander et al, 1997). They are meant to create a culture to document and

support architecture and design. They are reusable when similar problems arise and can be extended as solutions to new problems. They were transposed to computer science in the mid-nineties of the 20th century (Gamma et al, 1995; Coplien, 1996). Recent works (Staab et al, 2001; Blomqvist and Sandkuhl, 2005; Gangemi, 2005) propose the use of patterns for ontology engineering. We are however the first to propose patterns to help the design of ontology alignments.

If ontologies are designed using common patterns, or "best practices" methods, the engineering of alignments has much to benefit from the study of ontology design patterns. In (Clark et al, 2000), Clark *et al.* propose the use of *knowledge patterns* for engineering knowledge bases, using morphisms to instantiate the patterns. They particularly distinguish between the pattern, and implementation in a particular programming language. Staab *et al.* (Staab et al, 2001) propose the use of *semantic patterns* to engineer ontologies while remaining independent from the underlying ontology language. The separation between patterns as modeling tools and their implementation is directly related to the *Knowledge Level* paradigm (Newell, 1982), then applied to solve recurring reasoning problems (Clancey, 1985), and extended to general problem solving methods (Wielinga et al, 1992; Fensel et al, 2002). Our approach follows these works by explicitly separating ontology alignment patterns and grounded correspondences.

Gangemi (Gangemi, 2005) proposes the use of patterns based on experience and introduces a pattern template and a library of common patterns for ontology engineering.[4] Blomqvist *et al.* (Blomqvist and Sandkuhl, 2005) present a classification of the various kinds of patterns for ontology engineering, from fine grained patterns modeling simple elements in a particular knowledge representation formalism, to high-level patterns manipulating ontology modules or sets of ontologies for semantic applications design. Ontology alignment patterns introduced in this paper are situated at the *syntactic* and *semantic* levels of Blomqvist classification. Ontology engineering patterns are obviously closely related to ontology alignment patterns: patterns used to engineer ontology can be used to discover ontology alignment patterns. More recently an ontology patterns classification was proposed (Presutti and Gangemi, 2008b), including content (Presutti and Gangemi, 2008b), logical and ontology alignment patterns.

Research on ontology matching was given considerable attention over the past 15 years. A recent book relates the various techniques developed in this area (Euzenat et al, 2007). Research on schema matching (Rahm and Bernstein, 2001) strongly influences the field. The identification of mismatches and reconciliation techniques between two database schemas was first identified by Battini (Batini et al, 1986). This work lead to the construction of schema integration systems

[4]Note that one shouldn't mix patterns in knowledge engineering as in this paper, and patterns in data mining, as for example in (Salam and Khayal, 2012)

like Clio (Miller, 2007) and Rondo (Melnik et al, 2003). This last work studied grounding problems, and is studied as part of the model management field. There are a few mentions of patterns in the ontology matching area. In (Šváb, 2007), Šváb proposes to use patterns to obtain better automatic matching algorithms. In (Besana et al, 2005), Besana *et al.* developed a matching algorithm exploiting communication patterns in a peer-to-peer network. XML schema matching was also given considerable attention as recently in (Aïtelhadj et al, 2011). The work we present in this article is to the best of our knowledge the first attempt to formalize patterns for ontology alignment.

3 Ontology alignment patterns representation

Ontology alignment patterns have two dimensions. Conceptually, they are generic solutions to particular ontology alignment problems. Technically, they are integrated with semantic web standards for reuse in associated applications. We present in this section these two dimensions by giving a pattern template based on the usual pattern templates found in the literature. Technically, patterns are represented using an ontology alignment representation language.

3.1 A Pattern Template

A pattern template provides a standard way to represent patterns. They provide a simple format that can be used to express a pattern description. Following (Staab et al, 2001), we introduce an ontology alignment pattern template divided in two parts that correspond to the two lowest levels of Blomqvist pattern classification (Blomqvist and Sandkuhl, 2005). The first part, the core of the ontology alignment pattern, defines the pattern using classical elements from design patterns literature. The second part presents a grounding for that pattern in a knowledge representation formalism corresponding to a mediation task that can be solved by instanciating this pattern. Many groundings can be defined given a pattern.

The core part of the pattern template contains common pattern elements found in the literature. It follows the high level classification in (Gamma et al, 1995):

– NAME

Name A meaningful name to refer to the pattern.

Alias Nicknames and synonyms of the pattern name.

– PROBLEM

Problem A statement describing the intent, goals and objectives of the pattern.

Context The preconditions under which the problem recurs, the pattern's applicability.

– SOLUTION

Solution A description in natural language of the pattern. A description in an ontology alignment representation formalism.

Example instantiation Sample application of the pattern.

– CONSEQUENCES

Related Patterns Relationships between this pattern and others described using this template.

Known Uses Known occurrences of pattern instances.

Pattern template elements are implemented using the ontology design patterns annotation schema (Presutti et al, 2008). The grounding part represents the grounding of the instantiated pattern into a knowledge representation formalism or programming language of a mediator executing pattern instances. In order for an instantiated pattern to be grounded, there must be a grounding defined from the alignment representation formalism in which the pattern is expressed (see Section 3.2). The grounding part also specifies the mediation tasks for which this pattern can be used.

The ontology alignment patterns template grounding part includes the following elements:

Name of the target language/system The name of the formalism in which patterns instances are represented.

Applicability The possible extent of modelling the pattern in the given formalism.

Purpose The mediation task solved by the grounding. Can be for general purpose, query rewriting, instance transformation or ontology merging.

Example Grounding A representation of the pattern in the given formalism.

Comment Other information about the grounding.

Defining groundings from correspondences is in general ambiguous, this problem was notably investigated in (Popa et al, 2002). We present in the next section an ontology alignment representation language that will also allow us to define a precise representation of the correspondence modeled by each pattern.

3.2 Pattern Representation

Ontology alignment patterns are represented using the Expressive Declarative Ontology Alignment Language (EDOAL) (Euzenat et al, 2007).

EDOAL is an extension of the ontology alignment format (Euzenat, 2004). Its semantics allow to express alignments between ontologies modelled in various formalisms. This provides the possibility for ontology alignment patterns to be applied to various ontology languages, provided that these languages are managed at the tool level. EDOAL uses Classes, Relations, Properties, and Instances constructs to represent ontological entities. Each correspondence models a relation between the entities in the correspondence: equivalence, subsumption, disjointness, and membership of an individual to a class. Instances are individuals, Classes model classes of individuals, Relations are binary relations between individuals, Properties are binary relations standing between an individual and a data value. The term Attribute can be used when referring to a relation or a property without distinction. Expressive descriptions of correspondences can be modelled, allowing to capture the complex correspondences needed to solve ontology mismatches. Ontological entities can be grouped using set operators. Chains of relations and properties can be composed. Conditions and filters can be applied in order to restrict the scope of entities taking part in a correspondence. Transformation of property values is possible via the use of XPath functions. Finally we extend the language with variables allowing to represent placeholders for entities in patterns. Table 1 summarizes the representation constructs EDOAL provides.

Construct type	Construct name (abstract syntax)	Comment
Correspondence Relations	equivalence, subsumption, disjointness, membership ($\equiv, \sqsubseteq, \perp, \in$)	
Entities	Class, Relation, Property, Instance, value, datatype, comparator (C,R,P,u,Val,d,cp)	
class constructors	and,or,not ($\sqcap, \sqcup, \neg$)	set operators for class construction
relation constructors	and,or,not,inverse, transitive,reflexive,symmetric ($\sqcap, \sqcup, \neg$,inv(R),sym(R),trans(R),refl(R)) first, next ($R \cdot R$)	set operators for relation construction path relation constructors
properties constructors	and,or,not ($\sqcap, \sqcup, \neg$) first, next ($R \cdot P$) value, domain (cp(Val), dom(C), range(C))	set operators for properties construction path property constructors restrict the scope of a property according to its value, domain or range
property values transformation	transformation (transf(R))	transformation function on property values XPath transformations, or user defined
class restriction	attribute value ($\exists P$ cp Val), domain, cardinality ($\exists P$ cp d, $\exists \lvert P \rvert$ cp Val)	restrict the scope of a class according to the value of one of its attributes
variables	$?x$	placeholder for an entity

Table 1: EDOAL constructs summary

As generalized correspondences, ontology entities in patterns are represented using placeholders. In order to instantiate a pattern, placeholders are replaced with entity identifiers (URIs). The language has a RDF/XML syntax used for exchange of alignments across applications, and a more concised abstract syntax. The latter syntax will be used in Section 6 to present example instantiations of patterns on common ontology mismatches.[5]

There is a number of tools allowing to deal with alignments expressed in this formalism.[6] Matching algorithms are available as part of the Ontology Alignment Evaluation Initiative, outputting alignments in the alignment format.[7]

We are now able to present in detail the pattern solving the FOAF to vCard correspondence given as an introductory example.

3.3 Example patterns

We present below three patterns representative of various aspects of the pattern library (see Section 4). Patterns are given according to the template defined in the previous section.

The first pattern models a recurring correspondence when aligning schemas modeling data. The **Class by attribute value** pattern occurs when a class in one ontology corresponds to a class in the other ontology, given that the scope of the latter class is restricted to only those instances having a specific value for a given attribute. For example *AcceptedPaper* corresponds to *Paper* with *status* equals to *accepted*. The pattern example below specifies that the class *BordeauxWine* is equivalent to the class *Wine* with its attribute *terroir* restricted to the value *Bordeaux*.

Name: Class By Attribute Value Correspondence
URI: http://www.omwg.org/TR/d7/patterns-library#classByAttributeValue
Also Known As: classByPropertyValueMapping
Problem:
A class in one ontology is equivalent to part of a class in a second ontology. The scope of one class needs to be narrowed in order to fit the scope of the class in the other ontology.

Context: The scope of a class in one ontology is narrower than the scope of a class in the other ontology. The more general class has a property that can be used to narrow its scope in order to make it matching the scope of the narrower class.

[5]The language description is available at `http://alignapi.gforge.inria.fr/language.html`

[6]See for example the alignment API `alignapi.gforge.inria.fr/`

[7]OAEI: `http://oaei.ontologymatching.org`.

Solution:

Solution description:

This pattern establishes a correspondence between a class/property/property-value combination in one ontology and a class in another. This pattern is agnostic as to whether the correspondence is unidirectional or bidirectional. Direction of the correspondence can be achieved through combination of the pattern with the equivalentClassCorrespondence or subClassCorrespondence pattern.

Example instantiation:

```
<Cell>
 <entity1>
  <Class>
   <and>
    <Class rdf:about="vin:Vin">
   </and>
   <and>
    <AttributeValueRestriction>
     <onAttribute>
      <Property rdf:about="vin:terroir"/>
     </onAttribute>
     <comparator rdf:resource="&xsd;equal"/>
     <value>geo:Bordeaux</value>
    </AttributeValueRestriction>
   </and>
  </Class>
 </entity1>
 <entity2>
  <Class rdf:about="wine:BordeauxWine"/>
 </entity2>
 <relation>equivalent</relation>
</Cell>
```

Abstract Syntax:

vin:Vin $\sqcap\exists$ vin:terroir $=$ geo:Bordeaux $\equiv$ wine:BordeauxWine

Related Patterns: Equivalent Class Correspondence, Subclass Correspondence

Figure 3 illustrates the class by attribute value ontology alignment pattern presented above. Two classes are put into correspondence, and one class (on the right) is restricted by the value of one of its attributes. The example instantiation of Bordeaux wines is given in smaller font on the figure.

The following pattern solves the mismatch between FOAF and VCard persons presented in Section 1. We describe it here using the previously defined template. An illustration is given Figure 4.

Name: RCA-A Concat Pattern

Problem: An attribute in one ontology has the same intention as a class in another ontology. This class itself has many properties whose concatenated values form the value of the first ontology property.

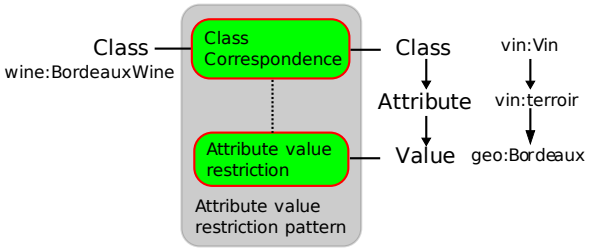

Figure 3: Class by attribute value pattern

Context: A class->property->value chain is present in one ontology, and a class->relation->class->property(ies) chain is present in the other ontology. The starting classes have a similar intent, and the ending properties model the same values.

Solution:

Solution description: This pattern establishes a correspondence between a relation, a class and one or many properties in one ontology, and a property in another ontology. The value of the property are determined by an aggregation function.

RDF/XML Syntax: See the instantiated example below.

Example instantiation:

```
<Alignment>
<map>
 <Cell>
  <entity1>
   <Relation rdf:about="vCard:name"/>
  </entity1>
  <entity2>
   <Property rdf:about="foaf:name"/>
  </entity2>
  <relation>equivalent</relation>
 </Cell>
</map>
 <map>
 <Cell>
  <entity1>
   <Property>
    <or>
     <Collection>
      <item>
       <Property rdf:about="vCard:family-name"/>
      </item>
      <item>
       <Property rdf:about="vCard:given-name"/>
      </item>
      <item>
       <Property rdf:about="vCard:additional-name"/>
      </item>
     </Collection>
    </or>
    <transf rdf:resource="transf:concat">
     vCard:given-name
     vCard:family-name
```

```
     vcard:additional-name
    </transf>
   </Property></entity1>
  <entity2><Property rdf:about="foaf:name"/></entity2>
  <relation>equivalent</relation>
 </Cell>
</map>
```

Abstract Syntax:
vCard:name $\equiv$ foaf:name
(vCard:family-name $\sqcup$ vCard:given-name $\sqcup$ vcard:additional-name) $\sqcap$ transf(concat, "vCard:family-name vCard:given-name vcard:additional-name") $\equiv$ foaf:name

Related Patterns: Equivalent Property Correspondence, Property Correspondence by Value

Figure 4 illustrates the relation class attribute to attribute concatenation ontology alignment pattern.

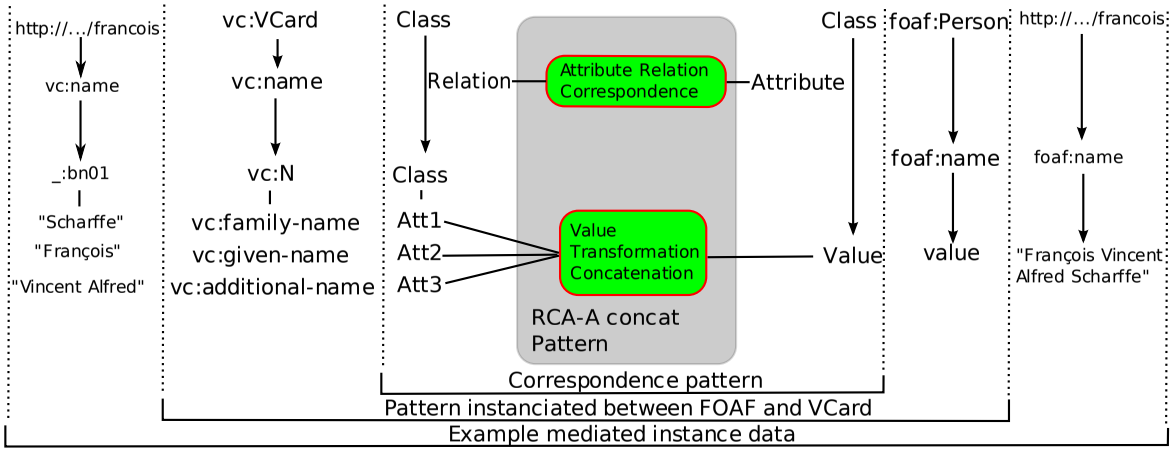

Figure 4: Relation Class Attribute to Attribute Concatenation Pattern

The third pattern illustrates a common correspondence needed when a property in one ontology is modeled as a relation in the other ontology. In an instance transformation scenario, data values of the property need to be transformed in instances of the class in the relation range. The grounding part of this pattern specifies it can be applied for this task.

Name: Property - Relation Correspondence

URI: http://www.omwg.org/TR/d7/patterns-library#propertyRelation

Alias: propertyRelationMapping

Problem: A property in one ontology corresponds to a relation in another ontology. The values of the property in the one ontology are expressed as individuals of a class for the relation in the other ontology.

Context: There is a correspondence between values of a property and individuals of a class.

Solution: *Solution description:* This pattern establishes an equivalent correspondence if the property range perfectly matches the individuals of the relation range. Otherwise a subsumption relation in one or the other direction is modeled.

RDF/XML Syntax: See example below.

Example instantiation:

```
<Cell>
 <entity1>
  <Property>
   <and>
    <Property rdf:about="o1:hasColor">
   </and>
   <transformation rdf:about="transf:colorsValuesToColorsInstances"/>
  </Property>
 </entity1>
 <entity2>
  <Relation rdf:about="o2:color"/>
 </entity2>
 <relation>subsumes</relation>
</Cell>
```

Abstract Syntax:

hasColor $\sqcap\exists$ transf(colorsValuesToColorsInstances) $\sqsubseteq$ color

Related Patterns: Equivalent Relation Correspondence, Equivalent Property Correspondence, Property-Relation Composite Pattern.

Figure 5 illustrates the property-relation pattern.

4 A pattern library

In the previous section we have given support to represent ontology alignment patterns using standard formalisms RDF and OWL. Based on this, we developed

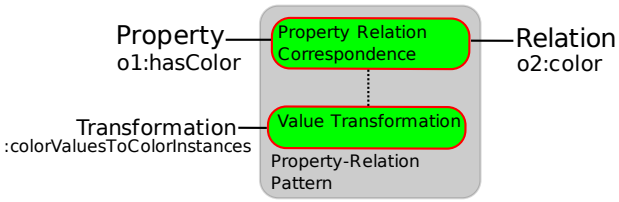

Figure 5: Property-relation pattern

a library of ontology alignment patterns. This library was constructed from two sources. First, we have conducted an empirical study based on an analysis of ontology mismatches (Scharffe, 2009). Then, the library is continually extended with new patterns through the ontology design patterns community portal (Presutti et al, 2008). A classification of the patterns in the library is given in Figure 6, which contains 49 patterns at the time of writing.

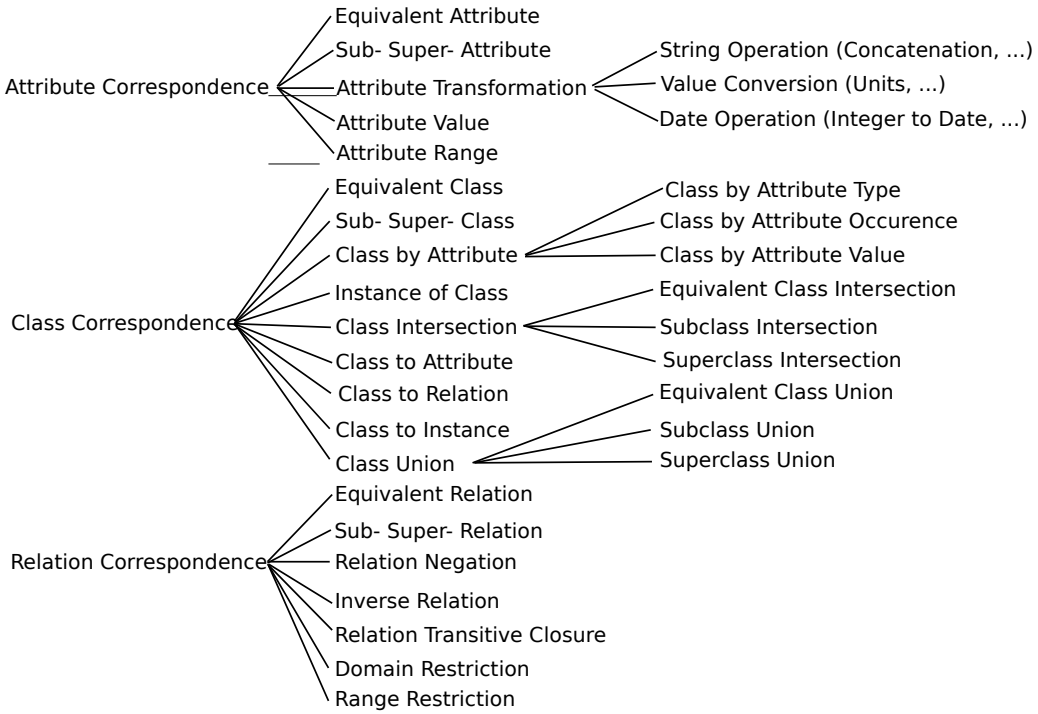

Figure 6: Pattern library

Different types of mismatches can occur when integrating heterogeneous ontologies (de Bruijn et al, 2006b). In order to correctly perform the alignment process, it is important to identify these mismatches as a first step towards proposing a solution for each of them. We have realized our study based on Klein's ontology mismatches classification (Klein, 2001), which is itself based on older classifica-

tions of Visser (Visser et al, 1997) and Wache (Wache, 2003), both mainly concerning discrepancies between database schemas. We also considered Euzenat's levels of interoperability (Euzenat, 2001) describing requirements for ontological interoperability.

Two general categories of mismatches are distinguished: mismatches at the ontology level including mismatches in the meaning or encoding of concepts in different ontologies, and mismatches at the language level concerning the underlying formalism used to model ontologies. Solving language mismatches requires work on the underlying logics of the ontology languages that are not in the scope of this paper. The following ontology mismatches can be distinguished:

Conceptualization mismatches appear when the concepts of a same domain is modeled differently. We distinguish here the *scope mismatches* appearing when two classes have some overlap in their extension. More precisely, a scope mismatch can reflect a *subsumption* between the two classes, an *incompatible scope* when the extensions have no overlap, a *categorization* mismatch when a different viewpoint is taken of a similar part of a domain, or an *aggregation level* mismatch reflecting a separation of the classes at different levels in the concept hierarchy.

Relation mismatches appear when the relations between concepts of a domain are modeled differently. A *structure mismatch* appears when relations have different intent though they can appear to relate equivalent entities in each ontology. An *attribute assignment* mismatch appears when an attribute is assigned at different levels in the concept hierarchy. An *attribute-type* mismatch arises when a property appearing in two ontologies has the same meaning, but its value range differs.

Model coverage and granularity mismatches come from differences in the part of the domain covered or the level of detail in which the domain is modeled.

Explication mismatches are related to differences in the description of the ontology entities. The *Paradigm* mismatch arises when different paradigms are used for the explication of a same concept. *Concept description* mismatches appear when different naming conventions are used to name entities. We distinguish two different mismatches here: *naming conventions* mismatches, and *terms based taxonomy* mismatches related to how concept names are constructed in a taxonomy. *Terminological* family of mismatches arise because of differences in the labels. These mismatches can be caused by the use of *synonym*, *homonym*, *hyponym* or *hypernym* relations between terms of the two ontologies.

Encoding mismatches arise when different data values in the two ontologies are encoded differently. A *Representation encoding* mismatch arises when two different units are used, a *data type* mismatch arises when different data types are used, and a *missing data* mismatch arises when a value is missing in one ontology because of weaker cardinality constraints.

45 patterns were identified in order to solve these ontology mismatches. The library is organized between more general and more specific patterns. Patterns are organized hierarchically according to their degree of generality. The deeper a pattern in the hierarchy, the more specialized for a specific problem. Specialization is reflected in the representation of the patterns using the alignment representation language. The top of the hierarchy is composed of the three basic patterns between classes, relations and attributes. Then each type of pattern is refined between equivalent, or subsumption patterns, and patterns specific for the entities types. These patterns will be further detailed in Section 6. For readability composite patterns associating many patterns are not shown on this figure. Part of the pattern library is published on the collaborative ontology design pattern portal: `http://ontologydesignpatterns.org`.

In the remaining of this article, we will show how patterns can be instantiated and used to solve ontology alignment problems. In Section 6 we evaluate the ability of patterns in the library to effectively model mismatches found in real world data. In Section 7 we present a system performing ontology transformations based on alignment patterns.

5 Applying ontology alignment patterns to solve ontology mismatches

In Section 4 we have introduced the pattern library as constructed based on mismatches arising when trying to align ontologies. We have then developed a framework allowing us to represent solutions to these mismatches using ontology alignment patterns expressed using the expressive and declarative ontology alignment language EDOAL. In this section, we will evaluate the possibility to solve every encountered mismatch using patterns from the pattern library. This evaluation only insures that patterns are available to solve known mismatches. Other types of mismatches might appear that will need to be formalized as new pattern using the proposed pattern template and the alignment language. For each mismatch we will try to find a relevant pattern solving it.

5.1 Methodology

For each mismatch, we go through the library of ontology alignment patterns and select a relevant one. We then instantiate the pattern with an example illustrating the various aspects of the mismatch. For each mismatch/example a small discussion explains how the mismatch is relevant to a certain ontology mediation task.

5.2 Ontology level mismatches

5.2.1 Conceptualization mismatches

Conceptualization mismatches are due to discrepancies between ontology structures. Solutions to these mismatches should thus take the ontologies structure into account to propose adapted correspondences.

I. Scope Mismatches Scope mismatches are solved by adapting the scopes of classes to fit them together. Class patterns will be used to solve this type of mismatch.

1. Subsumption mismatch The alignment of two classes, one being subsumed by the extension of the other does not raise any particular problem. The Class Subsumption pattern can be used to relate the two classes. In the following example, two classes Person are defined using a similar identifier, but the scope of the class in the first ontology is larger than the one in the second ontology.

$o_1 : Person \sqsubseteq o_2 : Person$

This pattern can easily be detected as a solution to the mismatch by a matcher (Euzenat and Shvaiko, 2007), see for example (Hu and Qu, 2008) comparing classes extensions.

2. Overlapping scope mismatch Depending on the ontology structure, the overlapping scopes mismatch can be solved using different strategies. In one case, attributes can be used in order to restrict classes scopes to the maximal intersection of their overlapping sets. This solution is the same for the categorization mismatch below. In the case of extension sets having a small size, instances can be directly aligned one-by-one. This is the case of the color example illustrating the solution below: only overlapping color instances are aligned. It is solved by using the Instance Equivalence pattern:

$o_1 : Blue \equiv o_2 : Blue \wedge o_1 : Green \equiv o_2 : Green \wedge o_1 : Red \equiv o_2 : Red$

When this solution is applied, instances will be considered equivalent. They can be related using an `owl:sameAs` statement.

3. Incompatible scopes mismatch The incompatible scope mismatch appears when instances of two classes in two ontologies have no overlap while the intention of the two classes is the same. This mismatch raises problems if an instance-based method is used to match the two ontologies. A simple equivalent classes pattern can be used to solve the mismatch, instantiated below on a Car example: the two instances sets of two ontologies with a Car class are disjoints, though the intension of the two classes are equivalent.

$o_1 : Car \equiv o_2 : Car$

This solution is valid for any mediation task. If the ontologies are to be merged, the merged extensions will fit together with no duplicates.

4. Categorization mismatch In order to align classes categorized differently, a composite pattern must be used. Each correspondence restricts the scope of a class in one ontology to the scope of a class in the other ontology, and each class in one ontology is given a correspondence to every class it has an overlap within the other ontology. The number of correspondences to be used is thus the number of classes in one ontology multiplied by the number of classes in the other ontology. The pattern fitting to this solution is the Class Scope Partition pattern instantiated below. In this example, one ontology distinguishes `mammals` and `birds` while the other ontology distinguishes `carnivores` and `herbivores`. Both categorizations represent a similar intension and can thus be aligned. :

$(o_1 : Mammal \sqcap \exists o_1 : foodHabits = carnivore) \equiv o_2 : Carnivore$
$(o_1 : Bird \sqcap \exists o_1 : foodHabits = carnivore) \equiv o_2 : Carnivore$
$(o_1 : Mammal \sqcap \exists o_1 : foodHabits = herbivore) \equiv o_2 : Herbivore$
$(o_1 : Bird \sqcap \exists o_1 : foodHabits = herbivore) \equiv o_2 : Herbivore$

These correspondences must also be designed by restricting the scopes of `Herbivore` and `Carnivore` using some of their attributes in order to match the scopes of `Mammal` and `Bird` and enable the execution of mediation tasks in the other direction.

5. Aggregation level mismatch In the aggregation level mismatch, a term in one ontology is more general than a term in another ontology because the latter ontology is more precise. The solution to this mismatch will be to either specialize the general term using a restriction (see the categorization mismatch above), or to use a subsumption relation from the more specific term to the more general (see the subsumption mismatch above).

II. Relation mismatches Relation mismatches are the following.

1. Structure Mismatch In the case of a structure mismatch, the domain and range of relations in the correspondence should be restricted so that they match together. In the given example, the relation `madeOf` ranges to a `Material` class and the relation `hasComponent` ranges to a `Component` class. As `Brick` can be as well a component and a material, ranges of both relations can be restricted to this class. A Relations By Range Restriction pattern can be used to relate the relations:

$o_1 : madeOf \sqcap range(o_1 : Brick) \equiv o_2 : hasComponent \sqcap range(o_2 : Brick)$

In case other components are as well considered as materials, the restriction could be relaxed by using the `or` class expression constructor and adding each relevant class into the expressions.

2. Attribute Assignment Mismatch In this mismatch, the domain of an attribute differs in the two ontologies. Depending if the attribute to be aligned is a property or a relation, a Property Domain Restriction pattern or a Relation Domain Restriction pattern can be used. In the example, two `color` properties are related, one being a color in general and the other a car color. We obtain the following solution:

$o_1 : color \sqcap dom(o1 : Car) \sqsupseteq o_2 : color$

3. Attribute-type mismatch In this mismatch, it is necessary to transform the value of a property from one type to another. The Property Value Transformation pattern gives the solution to this mismatch, it is below applied to transform miles to kilometers per hour between two `speed` properties:

$o_1 : speed \sqcap transf(MilesH2kmH) \equiv o_2 : speed$

While properties are semantically equivalent, the transfomation is useful for rewriting query values and tranforming instances from one ontology to the other.

III. Model coverage and granularity mismatches When one ontology has a finer granularity than another, some of its classes have no equivalent in the other ontology, but can be aligned as subclasses. The Subclass Correspondence pattern provides the solution to this mismatch, applied below to an example on cars, where the concept hierarchy of one ontology stops to `Car`, while the other also distinguishes `SportCar` and `CityCar`.
$o_1 : SportCar \sqsubseteq o_1 : Car$
$o_1 : CityCar \sqsubseteq o_1 : Car$

5.2.2 Explication Mismatches

I. Paradigm Mismatch The solution to this mismatch is very dependent on the paradigms involved in a particular case. The time example is a good illustration of the complexity of this mismatch. In one ontology, periods are indicated using strings, for instance `baroque`, `classical` and `contemporary`. In another ontology, periods are indicated using a start and an end date. In order to transform instances and queries from one ontology to the other, an external transformation function is needed that associates periods to dates. The Property Values Transformation pattern is here used two times to give the solution to this mismatch:
$o_1 : period \sqcap transf(period2startingDate) \equiv o_2 : startDate$
$o_1 : period \sqcap transf(period2endingDate) \equiv o_2 : endDate$

II. Concept Description Mismatches

1. Naming Convention This mismatch has much influence on matching systems. Knowing the naming convention allows to tune the matching system to make it more efficient. For example knowing that `has_` precedes every attribute can be used by the matcher to remove this prefix during the preprocessing phase. However, this mismatch has no effect on the way the solution is described.

2. Terms Based Taxonomy Here, and as in the Naming Convention mismatch, if the classes have different labels but same intension they can be aligned using the Equivalent Classes pattern. If this is not the case and the classes have different scopes this mismatch becomes a Scope Mismatch and takes the corresponding solution.

3. Element Partitioning This mismatch occurs when a set of properties in one ontology form a partition of a property in the other ontology. In the solution, the Property Partition By Value pattern allows to distinguish each partition, it is below applied on a name example similar to the name example between FOAF and vCard:

$(o_1 : firstName \sqcup o_1 : lastName) \sqcap$
$transf(concat,$"o1:firstName o1:lastName"$) \equiv o_2 : name$

III. Terminological Mismatches While synonym and hyponym mismatches are only relevant to matching algorithms and do not influence the representation of the subsequent correspondences, hypernym and hyponym terms will result in subsumbtion correspondences.

1. Synonym Synonym terms are a good indicator for a matcher that the entities are probably equivalent. An equivalence will result in a solution based on a Equivalent Classes, Equivalent Relations, or Equivalent Properties ontology alignment pattern. For example a class is named `Automobile` in one ontology and `Car` in the other ontology.

$o_1 : Automobile \equiv o_2 : Car$

2. Homonym In the case of homonym terms it is again relevant to the matcher, in a negative way this time. It will influence the matcher to have syntactically equal terms if the entities described by these terms are semantically different. The matcher can be helped if a correspondence is defined stating a disjointness between the terms. One of the disjoint entities pattern will be used to model the solution. Below the Disjoint Classes pattern is used to specify the two classes have no commonalities.

$o_1 : Top \perp o_2 : Top$

3. Hypernym & Hyponym terms If one term is the hypernym (resp. the hyponym) of the other this will increase matching algorithms confidence that there is a subsumption relation between concepts they label. If this is the case, a subsumption relation will be used to build the correspondence between the entities. Subsumption patterns thus provide a solution to these mismatches. The example below shows a subsumption between two classes `oak`, and `tree`:

$o_1 : Oak \sqsubseteq o_2 : Tree$

IV. Encoding Mismatches

1. Representation Encoding Mismatch The Representation Encoding mismatch is solved by using a Property Value Transformation pattern in order to convert from one unit to the other. This pattern is applied below to the given example aligning Kilograms with Pounds:

$o_1 : weight \sqcap transf(Kilograms2Pounds) \equiv o_2 : weight$

2. Datatype Encoding Mismatch Similar to the Representation Encoding mismatch, the Data Type Encoding mismatch is solved using a Data Value Transformation pattern. The pattern uses however the specific transformation corresponding to the datatypes to be converted. This pattern is applied below on the example of translation of an age property from a short value to an integer value. One can notice that this transformation is not always necessary as some systems will perform the translation automatically.

$o_1 : age \sqcap transf(Short2Integer) \equiv o_2 : age$

3. Missing Data Mismatch The Missing Data mismatch is solved by adding a constraint ensuring that instances have a value for the given property. The Attribute Occurence pattern is a solution to this mismatch. We apply it below for the `age` property on a `Person` class:

$o1 : Person \equiv o2 : Person \sqcap \exists |o2 : age| = 1$

We were able to solve every encountered mismatch by instantiating a pattern. This gives the pattern library a coverage making it usable for designing ontology alignments for semantic web applications. However, this does not mean the library is by any means complete. More patterns are expected to arise over time. The emergence of the semantic web will lead to the development of numerous ontologies. Aligning these ontologies will certainly bring new patterns. We particularly expect domain specific patterns to emerge.

In the remainder of this paper, we will perform the alignment of ontologies modelling linked-data and show that correspondence patterns are useful to design precise alignments. We will then present a system using alignments patterns for transforming ontologies. Conclusions wrap up the article.

6 Evaluation of Alignment Patterns

In order to evaluate alignment patterns we manually inspected 10 real-world examples from the linked data realm and we counted the number of occurrences of five alignment patterns. We used a subset of 10 examples from 42 examples from the Linked Open Data Benchmark (Rivero et al, 2012) where they serve for setting a real-world distribution of mapping patterns for a benchmark. Authors of (Rivero et al, 2012) focused on data sources from the Linked Open Data cloud[8] which have over 25.000 *owl:sameAs* links. Afterwards, they randomly selected 42 examples comprising a pair of instances connected by an *owl:sameAs* link. In our case we selected 10 examples so as to be representative in terms of occurrences of inspected alignment patterns. Therefore, we selected pairs of instances between DBpedia and Freebase and pairs of instances between DBpedia and LinkedGeoData datasets. While DBpedia and Freebase usually provide rich descriptions of instances from different domains and employ different vocabularies as well, LinkedGeoData is a very specific geolocation datasource. Consequently, there are very few similar structures with regard to other instances from multi-domains datasources.

In this experiment we focus on five different alignment patterns. First three alignment patterns are type of "Class by Attribute". The last two alignment patterns deal with value or datatype transformation:

[8] `http://thedatahub.org/group/lodcloud`

– "Class by Attribute Occurence" (CAO), establishes a mapping between a class and an attribute combination in one ontology and a class in another.

– "Class by Attribute Type" (CAT), establishes again a mapping between a class and an attribute combination in one ontology and a class in another, but in this case only those instances for which an attribute value corresponding to a certain type (subclass of the attribute range) is given are aligned.

– "Class by Attribute Value" (CAV), establishes a mapping between a class and a property-value combination in one ontology and a class in another.

– "Value Transformation" (VT), establishes a mapping between identical properties, however transformation of values is needed.

– "Datatype transformation" (DT) establishes a mapping between identical properties, however transformation of datatype is needed.

During the evaluation we manually counted the number of alignment patterns that should be applied in order to translate between different instances representations. For first three alignment patterns we analyzed both directions, as depicted along with resulting numbers in Table 2. Evaluation material is available online,[9] and we provide below one pattern instance example for each type of an alignment pattern.

Regarding CAO pattern instance example, the following instance[10]

```
@prefix fb: <http://rdf.freebase.com/ns/> .
fb:en.toledo_oregon a fb:location.citytown.
fb:en.toledo_oregon fb:location.statistical_region.population [].
```

can be translated into the following instance

```
@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .
@prefix dbpedia: <http://dbpedia.org/resource/> .
dbpedia:Toledo,_Oregon a dbpedia-owl:PopulatedPlace.
```

using the CAO alignment pattern:[11]

fb:location.citytown $\sqcap\exists$ fb:location.statistical_region.population $\equiv$
dbpedia-owl:PopulatedPlace.

There are no occurrences in the case of CAT pattern. This is caused by the fact that we evaluate these alignment patterns on linked data where we directly

[9] All data for this experiment such as rdf or n3 dumps for each example, statistics and alignment patterns instances are available at: `http://nb.vse.cz/~svabo/KAIS2012/`

[10] For the sake of brevity we only place square brackets.

[11] For the sake of brevity we use abstract syntax.

direction	→			←			↔	
alignment pattern example	CAO	CAT	CAV	CAO	CAT	CAV	VT	DT
1	1	0	0	1	0	2	1	1
2	0	0	1	0	0	2	1	1
3	0	0	7	0	0	0	0	1
4	1	0	0	0	0	1	1	1
5	0	0	0	0	0	6	1	2
6	0	0	0	0	0	4	1	1
7	0	0	0	0	0	0	0	4
8	0	0	0	0	0	0	0	3
9	0	0	0	0	0	1	0	3
10	0	0	0	0	0	1	0	3

Table 2: Numbers of occurrences of five alignment patterns in 10 real-world linked data examples.

see instances instead of classes. We can state one example which is very close to CAT pattern. Let us consider translating the following instance:[12]

```
@prefix fb: <http://rdf.freebase.com/ns/> .
fb:en.berlin a fb:location.citytown.
fb:en.berlin fb:location.administrative_division.country fb:en.germany.
```

to the following instance

```
@prefix yago: <http://dbpedia.org/class/yago/> .
@prefix dbpedia: <http://dbpedia.org/resource/> .
dbpedia:Berlin a yago:StatesOfGermany.
```

$fb : en.germany$ is actually instance. However, if this was a class with instances like “Hesse”, “Bavaria” etc. Then, related CAT pattern would look like:

fb:location.citytown $\sqcap\exists$ fb:location.administrative_division.country $=$ fb:en.germany $\equiv$ yago:StatesOfGermany.

CAV pattern is the most frequent one. We can say that it is quite typical for linked data. For example, let us consider translating the following instance:

```
@prefix fb: <http://rdf.freebase.com/ns/> .
fb:en.berlin a fb:location.citytown.
fb:en.berlin fb:location.dated_location.date_founded 1237.
```

[12]In our analysis we considered not only entities from DBpedia schema but also others which were present in an DBpedia instance description, e.g. yago ontology.

to the following instance

```
@prefix yago: <http://dbpedia.org/class/yago/> .
@prefix dbpedia: <http://dbpedia.org/resource/> .
dbpedia:Berlin a yago:PopulatedPlacesEstablishedInThe13thCentury.
```

For such a transformation, we can apply the following CAV pattern:

fb:location.citytown $\sqcap\exists$ fb:location.dated_location.date_founded $=$ 1237 $\equiv$ yago:PopulatedPlacesEstablishedInThe13thCentury.

VT pattern can be useful in many situations, e.g. let us consider that we want to transform the following instance:

```
@prefix fb: <http://rdf.freebase.com/ns/> .
fb:en.the_godfather fb:film.film_cut.runtime 175.
```

to the following instance

```
@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .
@prefix dbpedia: <http://dbpedia.org/resource/> .
dbpedia:The_Godfather dbpedia-owl:runtime 10500.
```

For such a transformation, we can apply the following VT pattern:

fb:film.film_cut.runtime $\sqcap$ transf(Minutes2Seconds) $\equiv$ dbpedia-owl:runtime.

Finally, to illustrate DT pattern let us consider a transformation of the following instance:

```
@prefix fb: <http://rdf.freebase.com/ns/>.
fb:en.clint_eastwood fb:people.person.date_of_birth 1930-05-31.
```

to the following instance

```
@prefix dbpprop: <http://dbpedia.org/property/>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
dbpedia:Clint_Eastwood dbpprop:birthDate "1930-05-31"^^xsd:date.
```

For such a transformation, we can apply the following DT pattern:

fb:people.person.date_of_birth $\sqcap$ transf(Literal2Date) $\equiv$ dbpprop:birthDate.

By using alignment patterns as shown above we can increase precision of data translation. For instance, using the CAV pattern in the case of yago : PopulatedPlacesEstablishedInThe13thCentury example we can translate all towns (instances) from Freebase dataset which are founded in 1237 (in fact there could be year interval for the whole century). If we only have an equivalence match between towns we would not have a translation/classification stating e.g. that 'Rietberg is an instance of yago:PopulatedPlacesEstablishedInThe13thCentury' which would decrease a precision of such a translation between datasets. To sum up, this

evaluation shows that alignment patterns are helpful in a data translation scenario. While we did not discover any occurrence of the CAT pattern, there were more CAV patterns applicable in the case more generic pairs of data sources were taken into account. On the other hand, for transforming representation the case of a more general data source on the one side and a very specific one on the other side does mainly need the DT pattern. This is a typical situation from a linked data perspective.

7 Transformation of ontologies using ontology alignment patterns

This approach (Šváb-Zamazal et al, 2009)[13] aims for pattern-based ontology transformation. It consists in detecting and transforming ontology fragments corresponding to patterns. Transforming ontologies acording to a given transformation pattern can find useful applications in a variety of scenarios where one needs to refactor an ontology. The approach workflow is given in Figure 7.

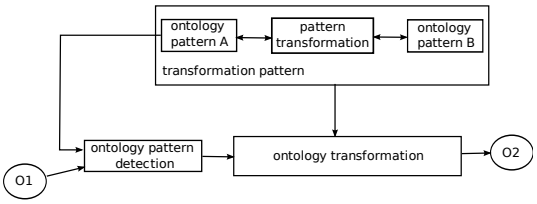

Figure 7: Pattern-based transformation workflow

A "transformation pattern" contains a source pattern to be detected, a target pattern, and a pattern tranformation modeling the correspondence between the two ontology patterns. If the source pattern is detected in an ontology, transformation instructions are generated according to correspondences between entities from source pattern and target pattern. Finally, transformation is applied following transformation instructions. In this application, ontology alignment patterns are used as a basis for transformation patterns, particularly for pattern transformation part.

Let's take the example of the transformation pattern related to 'Class by Attribute Type pattern'.[14] Source pattern also consists of naming detection pattern (ndp) in order to increase precision of detection. Instead of using specified entities, there are placeholders which are bound during pattern detection phase:

[13]Tool is available at: `http://owl.vse.cz:8080/`

[14]This is based on online example dealing with ontology matching use case available at: `http://owl.vse.cz:8080/tutorial/node37.html`

```
<op1>
   <entity_declarations>
      <placeholder type="ObjectProperty">?p</placeholder>
      <placeholder type="Class">?A</placeholder>
      <placeholder type="Class">?B</placeholder>
      <placeholder type="Class">?C</placeholder>
   </entity_declarations>
   <axioms>
      <axiom>?p domain ?A</axiom>
      <axiom>?p range ?B</axiom>
      <axiom>?C subClassOf ?B</axiom>
   </axioms>
   <ndp>
     <comparison threshold="1.0" measure="equal">
      <s1>?B</s1>
      <s2>head_term(?p)</s2>
     </comparison>
     <exist>verb_form(?C)</exist>
   </ndp>
</op1>
```

Target pattern consists of new entity ?G:

```
<op2>
  <entity_declarations>
    <placeholder type="ObjectProperty">?q</placeholder>
    <placeholder type="Class">?D</placeholder>
    <placeholder type="Class">?E</placeholder>
    <placeholder type="Class">?F</placeholder>
    <placeholder type="Class">?G</placeholder>
  </entity_declarations>
  <axioms>
    <axiom>?q domain ?D</axiom>
    <axiom>?q range ?E</axiom>
    <axiom>?F subClassOf ?E</axiom>
    <axiom>?G equivalentTo (?q some ?F)</axiom>
    <axiom>?G equivalentTo ?D</axiom>
    <axiom>?G subClassOf ?D</axiom>
  </axioms>
</op2>
```

Finally, there is pattern transformation which link entities from both ontology patterns. There is also information how to name new entity (?G) in ntp (naming transformation pattern) element.

```
 <pt>
    <eq op1="?A" op2="?D"/>
    <eq op1="?B" op2="?E"/>
    <eq op1="?C" op2="?F"/>
    <eq op1="?p" op2="?q"/>
    <ntp entity="?G">make_passive_verb(?C)+head_noun(?A)</ntp>
    <ntp entity="?D">?A</ntp>
    <ntp entity="?E">?B</ntp>
    <ntp entity="?q">?p</ntp>
  </pt>
```

This transformation pattern is based on alignment pattern “Class by attribute type”:

```
<Cell>
<entity1>
  <Class>
   <and>
    <Class rdf:about="?A"/>
   </and>
   <and>
    <AttributeTypeCondition>
     <onAttribute>
      <Relation rdf:about="?p"/>
     </onAttribute>
     <value><Instance rdf:about="?C"></value>
    </AttributeTypeCondition>
   </and>
  </Class>
 </entity1>
 <entity2>
  <Class rdf:about="?G"/>
 </entity2>
</Cell>
```

For example, concrete final complex alignment would be with the following binding of placeholders: $?A$=Paper, $?p$=hasDecision, $?C$=Acceptance and $?G$=AcceptedPaper.

The general goal of this approach is to offer an ontology transformation service enabling to adapt ontologies for specific scenarios. We have performed experiments on improving the quality of ontology matching by preprocessing ontologies with transformations (Šváb-Zamazal et al, 2009).

8 Conclusion

In this article we have introduced ontology alignment patterns as a means to formalize solutions to ontology mismatches. This is to our knowledge the first attempt to identify and formalize alignment patterns. We have positioned ontology alignment patterns in the ontology mediation landscape as abstract representations of correspondences between ontology entities. We have formalized patterns by giving them a template and a representation based on the expressive and declarative ontology alignment language, making ontology alignment patterns available for practical mediation tasks. We were able to solve every recognized ontology mismatch by instantiating a pattern from the proposed pattern library. We have presented an application using ontology alignment patterns for ontology transformation.

The approach presented in this article aims at providing patterns between ontologies reusable for various purposes. We have designed these patterns following the semantic web framework and languages. They are thus generic inside this framework, the alignment language allowing to align ontologies designed in

RDFS, or OWL variants. However, other knowledge representation languages can have a different underlying model. Our approach is in that sense limited, ontology alignment patterns are not meant to provide translation between heterogeneous ontology languages, ie to solve language mismatches.

Another limitation of the pattern representation formalism is to have a fixed argument size for entities aggregation and value transformations. We cannot at the current stage represent patterns with a variable number of entities at a certain position. The concatenation of many property values presented in the article is an example of such a fixed argument size. Extension of the formalism to support variable arguments size is on our research agenda.

It can be argued that the usage of patterns in applications is highly dependant on the availability of groundings in the mediator languages of these applications. Moreover, providing such groundings is not trivial. The work realized as part of the alignment representation language and the alignment API is going in the direction of providing more support for groundings.

When it comes to the usage of patterns by the user, a graphical interface associated with pattern detection and other functionalities like the transformation service we propose would be of great help. Such an interface would also give more evidence that pattern-based ontology alignment design makes the task easier for the user.

Acknowledgements Ondřej Zamazal has been partially supported by the CSF grant no. P202/10/1825.

References

Aïtelhadj A, Boughanem M, Mezghiche M, Souam F (2011) Using structural similarity for clustering XML documents. In *Knowledge and Information Systems* 32(1): 109–139

Alexander Ch, Ishikawa S, Silverstein M (1977) A Pattern Language. Oxford University Press, New York, USA, pp 1216

Batini C, Lenzerini M, Navathe S B (1986) A comparative analysis of methodologies for database schema integration. In *ACM Computing Surveys* 18(4): 323–364

Besana P, Robertson D, Rovatsos M (2005) Exploiting interaction contexts in P2P ontology mapping. In International Workshop on Peer-to-Peer Knowledge Management (P2PKM), San Diego, CA, USA, July 2005

Bizer C (2003) D2R MAP - A Database to RDF Mapping Language. In Proceedings of the 12th International World Wide Web Conference (Posters), Budapest, May 2003

Blomqvist E, Sandkuhl K (2005) Patterns in Ontology Engineering: Classification of Ontology Patterns. In Proceedings of International Conference on Enterprise Information Systems (3): 413–416

Chiticariu L, Hernández M A, Kolaitis P G, Popa L (2007) Semi-automatic schema integration in Clio. In Proceedings of the 33rd international conference on Very large data bases (VLDB '07), Vienna, Austria, pp 1326–1329

Clancey W J (1985) Heuristic classification. In *Artificial Intelligence* 27(3): 289–350

Clark P, Thompson J, Porter B (2000) Knowledge Patterns. In Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR2000), pp 591–600

Coplien J O (1996) Software Patterns. SIGS Books, New York

de Bruijn J, Ehrig M, Feier C, Martin-Recuerda F, Scharffe F, Weiten M (2006), Ontology Mediation, Merging, and Aligning. In Davies J, Studer R, Warren P (eds) Semantic Web Technologies. John Wiley & Sons, Ltd.

Doan A, Halevy A Y (2005) Semantic-integration research in the database community. In *AI Magazine* 26(1): 83–94

Dorneles C F, Gonçalves R, dos Santos Mello R (2011) Approximate data instance matching: a survey. In *Knowledge and Information Systems* 27(1):1–21

Euzenat J (2001) Towards a principled approach to semantic interoperability. In Gomez-Perez A, Gruninger M, Stuckenschmidt H, Uschold M (eds). In Proceedings of Workshop on Ontologies and Information Sharing, IJCAI'01

Euzenat J (2004) An API for Ontology Alignment. In van Harmelen F, McIlraith S, Plexousakis D (eds). In Proceedings of the 3rd International Semantic Web Conference, Hiroshima, Japan, pp 698–712

Euzenat J, Scharffe F, Zimmermann A (2007) D2.2.10: Expressive alignment language and implementation. In project deliverable 2.2.10, Knowledge Web NoE (FP6-507482)

Euzenat J, Shvaiko P (2007) Ontology Matching. In Springer-Verlag, Heidelberg (DE), pp 341

Falconer S M, Storey M-A D (2007) A Cognitive Support Framework for Ontology Mapping. In Proceedings of ISWC/ASWC, pp 114–127

Fensel D, Motta E, Benjamins V R Crubezy M, Decker S, Gaspari M, Groenboom R, Grosso W, van Harmelen F, Musen M, Plaza E, Schreiber G, Studer R, Wielinga B (2002) The Unified Problem-solving Method Development Language UPML. In *Knowledge and Information Systems* 5(1): 83–131

Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA

Gangemi A (2005) Ontology Design Patterns for Semantic Web Content. In Proceedings of the 4th International Semantic Web Conference, pp 262–276

Ghazvinian A, Noy N F, Jonquet C, Shah N H, Musen N A (2009) What Four Million Mappings Can Tell You About Two Hundred Ontologies. In Proceedings of the ISWC 2009, Springer

Groza T, Grimnes G A A, Handschuh S, Decker S (2011) From raw publications to Linked Data. In *Knowledge and Information Systems*, pp 1–21

Gruber T R (1995) Toward principles for the design of ontologies used for knowledge sharing. In *International Journal of Human-Computer Studies* 43(4-5): 907–928

Hu W, Qu Y (2008) Falcon-AO: A practical ontology matching system. In *Journal of Web Semantics* 6(3): 237-239

Kalfoglou Y, Schorlemmer M (2003) Ontology mapping: the state of the art. In *The Knowledge Engineering Review* 18(1): 1–31

Klein M (2001) Combining and relating ontologies: an analysis of problems and solutions. In Proceedings of workshop on Ontologies and Information Sharing

McGuinness D L, Fikes R, Rice J, Wilder S (2000) The Chimaera Ontology Environment. In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000)

Melnik S, Rahm E, Bernstein P A (2003) Rondo: A Programming Platform for Generic Model Management. In Proceedings of SIGMOD 03

Miller R J (2007) Retrospective on Clio: Schema Mapping and Data Exchange in Practice. In Proceedings of the 2007 International Workshop on Description Logics (DL2007), Brixen-Bressanone, Italy

Mocan A, Cimpian E, Kerrigan M (2006) Formal Model for Ontology Mapping Creation. In Proceedings of the International Semantic Web Conferenc (ISWC 2006), pp 459–472

Newell A (1982) The Knowledge Level. In *Artificial Intelligence* 18(1):87-127

Noy N F, Griffith N, Musen M A (2008) Collecting Community-Based Mappings in an Ontology Repository. In Proceedings of International Semantic Web Conference (2008), pp 371-386

Noy N F, Musen M A (2000) PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pp 450–455

Oztemel E, Arslankaya S (2012) Enterprise knowledge management model: a knowledge tower. In *Knowledge and Information Systems* 31(1):171–192

Le Phuoc D, Polleres A, Morbidoni C, Hauswirth M, Tummarello G (2009) Rapid semantic web mashup development through semantic web pipes. In Proceedings of the 18th World Wide Web Conference (WWW2009), Madrid, Spain, pp 581–590

Popa L, Velegrakis Y, Miller R J, Hernández M A, Fagin R (2002) Translating Web Data. In Proceedings of International Conference on Very Large Data Bases (VLDB), pp 598–609

Presutti V, Daga E, Gangemi A, Salvati A (2008), http: //ontologydesignpatterns.org [ODP]. In Posters & Demos Session of the 7th International Semantic Web Conference (ISWC 2008)

Presutti V, Gangemi A (2008) Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies. In Proceedings of International Conference on Conceptual Modeling (ER), pp 128-141

Rahm E, Bernstein P A (2001) A survey of approaches to automatic schema matching. In *The VLDB Journal* 10(4):334–350

Rivero C R, Schultz A, Bizer C, Ruiz D (2012) Benchmarking the Performance of Linked Data Translation Systems. In Proceedings of Linked Data on the web (LDOW2012)

Salam A, Khayal M S H (2012) Mining top-k frequent patterns without minimum support threshold. In *Knowledge and information systems* 30(1):57–86

Scharffe F (2009) Correspondence Patterns Representation. Ph.D Thesis, University of Innsbruck, 2009

Shvaiko P, Euzenat J (2005) A Survey of Schema-Based Matching Approaches. In *Journal on Data Semantics* IV, pp 146–171

Shvaiko P, Euzenat J (2008) Ten Challenges for Ontology Matching. In Proceedings of The 7th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2008), pp 1164–1182

Staab S, Erdmann M, Maedche A (2001) Engineering ontologies using semantic patterns. In Proceedings of the IJCAI-01 Workshop on E-Business & the Intelligent Web, Seattle, WA, USA, August 5, 2001

Šváb O (2007) Exploiting patterns in Ontology Mapping. In Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea, Springer Verlag, pp 950–954

Šváb-Zamazal O, Svátek V, Scharffe F (2009) Pattern-based ontology transformation service. In Proceedings of the first International Conference on Knowledge Engineering and Ontology Development (KEOD'09), Madeira, Portugal

Ullman J D (1997) Information Integration Using Logical Views. In Proceedings of The 11th International Conference on Database Theory (ICDT 2007), pp 19-40

Visser P R S, Jones D M, Bench-Capon T J M, Shave M J R (1997) An Analysis of Ontological Mismatches: Heterogeneity versus Interoperability. In AAAI 1997 Spring Symposium on Ontological Engineering, Stanford, USA, pp 164–172

Wache H (2003) Semantische Mediation für heterogene Informationsquellen. In *Journal of Kuenstliche Intelligenz* 17(4)

Wielinga B J, Schreiber A T, Breuker J A (1992) KADS: a modelling approach to knowledge engineering. In *Journal of Knowledge Acquisition* 4(1):5–53, Academic Press Ltd., London, UK

Zhdanova A V, Shvaiko P (2006) Community-Driven Ontology Matching. In Proceedings of the 3rd European Semantic Web Conference (ESWC 2006), pp 34–49