Towards Design Patterns for Ontology Alignment*

François Scharffe
University of Innsbruck
Austria
francois.scharffe@uibk.ac.at

Jérôme Euzenat
INRIA
Grenoble, France
jerome.euzenat@inrialpes.fr

Dieter Fensel
University of Innsbruck
Austria
dieter.fensel@uibk.ac.at

ABSTRACT

Aligning ontologies is a crucial and tedious task. Matching algorithms and tools provide support to facilitate the task of the user in defining correspondences between ontologies entities. However, automatic matching is actually limited to the detection of simple one to one correspondences to be further refined by the user. We propose in this paper the use of correspondence patterns as a tool to assist the design of ontology alignments. Based on existing research on patterns in the fields of software and ontology engineering, we propose a pattern template as an helper to develop a correspondence patterns library. We give ways towards the representation of patterns using an appropriate correspondence representation formalism: the Alignment Ontology.

1. INTRODUCTION

Far from the ideal world envisioned in early ontology research the semantic web contains many ontologies, and is expected to contain more and more as it will develop. When two ontologies overlaps, they can be linked together in order to enable exchange of their underlying knowledge. An *alignment* between two ontologies specifies a set of *correspondences*, and each correspondence models a bridge between a set of ontologies entities. Designing ontology alignments is a tedious task. There are many ongoing efforts to develop tools such as graphical user interfaces and matching algorithms, in order to make it easier.

Most ontology matching systems are limited to detect simple equivalence or subsumption correspondences between single entities, and research concentrates on improving their quality on various datasets more than on finding more complex matches[1]. This can be explained as the problem of detecting complex matches is not a trivial one, maybe also because there is no standard rule language on the semantic web.

However, simple correspondences are often not sufficient to correctly represent the relationship between the aligned entities. The two ontologies in the following example deal with wines. In the "Wine Ontology"[2], which main class is $Wine$, the class $BordeauxWine$ represents instances of a popular french wine made in the region around Bordeaux. In the "Ontologie du Vin"[3] a similar wine is expressed as an instance of Vin with an attribute $terroir$[4] indicating the wine provenance. Matching systems are able to detect the two correspondences $Wine \equiv Vin$ stating equivalence between two wines and $BordeauxWine \sqsubseteq Vin$ stating that $BordeauxWine$ is a narrower concept than Vin. In this case, a more precise correspondence would be $BordeauxWine \equiv Vin \sqcap \exists terroir.\{Bordeaux\}$, restricting the scope of Vin to only those instances having "$Bordeaux$" as value of the *terroir* attribute.

Going from the initial two basic correspondences to the refined one involving a condition would be easier using a pattern showing the structure of the correspondence.

Inspired from design patterns in software engineering, this paper introduces correspondence patterns as helpers that facilitate the ontology alignment process. They improve graphical tools by assisting the user when creating complex correspondences, and we believe they are a first step towards matching algorithms able to detect complex matches. We base the pattern representation on a correspondence representation formalism modeling correspondences at an abstract level. This allows to later on retrieve and use the correspondence according to the needs.

1.1 Knowledge Level

We consider three possible levels of abstraction when representing correspondences between ontologies. We find at the bottom of the stack *grounded correspondences*, as represented in a mediation system executing them. Classical representations of correspondences at this level are the logical formalism of the reasoner, or the programming language of the mediator system. For example a transformations of instances from the class *source* of an source ontology o_s to

*The work presented in this paper has been supported by the European Commission under the projects SUPER (FP6-026850), Knowledge Web (FP6-507482) and SEKT (IST-2003-506826)

[1]See the Ontology Alignment Evaluation Initiative 2007 evaluation criteria: http://oaei.ontologymatching.org/

[2]The Wine Ontology is available at: `http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine`.

[3]The Ontologie du Vin is available at: `http://www.scharffe.fr/ontologies/OntologyDuVin.wsml`.

[4]Terroir is a French word for a particular agricultural region

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

the class *target* of a target ontology o_t is represented in the following SPARQL query:

CONSTRUCT{ ?X rdf:type o_s:source }
WHERE{ ?X rdf:type o_t:target. }

The query above constitutes an example of grounded correspondence for the specific task of instance transformation. Other mediation tasks such as query rewriting or ontology merging require different systems to be performed. However, the correspondence between related entities remains the same at more abstract level.

We therefore introduce the *correspondence* level where correspondences are represented in a way they can be grounded to the language of the mediator system associated to the mediation task to be realized. This allows to specify correspondences independently from the expressiveness of the underlying knowledge representation formalism of the ontologies. Correspondences at this level can be conveniently exchanged between graphical tools, matchers and mediators. Correspondences are typically expressed using a specific ontology alignment representation formalism such as the Alignment Ontology presented Section 1.2.

A *correspondence pattern* is at the top level of abstraction of the ontology alignment representation stack. Correspondence patterns are essentially correspondences and sets of correspondences with unspecified entities. They act as templates to help finding more precise correspondences than simply relating one entity to another. Correspondence patterns are meant to be used in graphical user interfaces, in order to refine simple correspondences discovered by matching algorithms.

Previous and ongoing works in ontology mediation focus on the development of tools and representation formalisms for ontology alignment. Grounded correspondences are often expressed as logical axioms, like in C-OWL [5] or Distributed Description Logics [4]. These works however do not differentiate the ontology specification from the alignment, making more difficult to maintain alignments as the ontology evolve. They also propose rules in a particular reasoning scheme that is not necessarily reusable for other tasks. For example, it is not possible to merge ontologies based on C-OWL bridge rules. Abstraction from the ontology language appear in ontology alignment specific formalisms. The Bridge Ontology in the mapping framework MAFRA [14], the Alignment Ontology and its serializations [10] are such formalisms allowing to specify correspondences between ontologies at an abstract level. We will in the following present the Alignment Ontology as it will later be used as the basis to develop the pattern library presented Section 3.

1.2 Representing Correspondences

The Alignment Ontology is an OWL-DL ontology that models ontology alignments as sets of correspondences between ontological entities. It results from efforts [9, 19] to create an ontology alignment format, abstract from the ontology language, that could serve as an interchange format between matching algorithms, mediators and graphical user interfaces.

Main classes of the ontology are `Alignment` which represents a set of correspondences between ontologies, and `Cell` which represents a single correspondence. Four ontological entities are considered: `Class`, `Attribute`, `Relation`, and `Instance`. Simple correspondences can be specified between single entities, more elaborated correspondences can be constructed between multiple entities using set operators. Conditions restrict entities scope by regarding the type, occurrence or value of an attribute, but also by regarding the domain or range of relations. Transformations of attributes values can also be specified using XPath functions[5]. A `measure` property indicates a confidence measure in the correspondence. A `relation` indicates the relation standing between the related entities: equivalence or subsumption. A model-theoretic semantics for the alignment representation formalism modeled by the Alignment Ontology is given in [10]. The ontology is available at [18].

This paper is organized as follows: in Section 2 we present a pattern template to model correspondences, based on the related work on design patterns. In Section 3 we propose to use this template to develop a correspondence patterns library in a semantic web setting.

2. A PATTERN TEMPLATE

We develop in this section a pattern template [8, 20] providing a standard way to represent patterns. The template is divided in two parts, the core part of the pattern template contains common pattern elements found in the literature. It follows the high level classification of the GoF [11]. Patterns modeled using this template represent the information needed to guide a user trying to find the relevant pattern for the matching problem to be solved.

The second part presents a grounding for that pattern in a knowledge representation formalism. This separation in two parts allows the definition of many groundings given a pattern such as suggested in [21].

We give in the following a detailed presentation of the template elements.

A Correspondence Pattern represents a correspondence between two ontologies.

- NAME
  - Name: A meaningful name to refer to the pattern. The name is used to refer to the pattern in the system it is developed. In a semantic web setting, the name is usually a fragment of the URI identifying the pattern.
  - Alias: Nicknames and synonyms of the pattern name. When different versions of a pattern exist, the name my as well change and the old name still referred by the Alias.
- PROBLEM
  - Problem: A statement describing the intent, goals and objectives of the pattern. This element describe how a certain correspondence, is modeled using this pattern and what are the entities involved by the pattern.
  - Context: The conditions under which the pattern happens to be useful. This element refer to specific domains where the pattern is relevant if any.
- SOLUTION

[5]XPath functions specification, W3C recommendation available at `http://www.w3.org/TR/xpath-functions/`

- Solution: A descriptions in natural language of the pattern, which entities are related to which entities, conditions and transformations used if any. A description as an example instance of the Alignment Ontology.
- Examples: A sample application of the pattern. The example is meant to give a clearer idea of when and how the pattern can be applied.

- CONSEQUENCES
  - Related Patterns: Relationships between this pattern and others described using this template. This element refer to similar patterns with one varying parameter, as well as refined or more general versions of the pattern.
  - Known Uses: Known occurrence of pattern instances and lessons learned. This is where the community aspect of patterns comes into play. Each time the pattern is successfully used to design a correspondence between two ontologies, this element should refer to the created correspondence.

Built on this template, the following pattern shown Table 2 can be used to model the correspondence presented in Section 1.

As we can see in this example, ontological entities in the patterns are given generic names (*ClassWithCondition, RestrictionInstance, TargetClass, &omwg;nil*). The patterns library defined Section 3 should provides such generic entities.

With regards to the correspondence representation stack introduced Section 1.1, the second part of the pattern is used to obtain a grounded correspondence from a pattern instance. This part represents the grounding of the pattern into a knowledge representation formalism used by a mediator executing pattern instances. The main purpose of the grounding part of the pattern is to give an indication to the user of the possibility to apply the pattern in the system he is working with. For example, a pattern involving a condition or a data transformation cannot be used in a system based on a DL-reasoner as the expressiveness of this language is not sufficient.

We will see next in Section 3 how correspondence patterns can be encoded in a machine readable format in order to facilitate their use in semantic web applications and we give a first set of generic patterns we have identified.

3. CORRESPONDENCE PATTERNS FOR THE SEMANTIC WEB

Based on the template given in the previous section, we developed a number of generic patterns answering common mismatches appearing when trying to relate ontologies, according to the description of ontology mismatches in [13]. The pattern library is organized in an OWL class hierarchy, extending the Alignment Ontology [18]. In fact, patterns are represented as specific kind of correspondences whose possible instantiations are restricted using OWL axioms.

The class hierarchy automatically organizes patterns according to their specificity, and the degenerated version of

Name: Class By Attribute Correspondence
Alias: classByAttributeCorrespondence
classByAttributeMapping

Problem:
A class in one ontology is related to a class in the other ontology. The scope of the class of the first ontology is restricted to only those instances having a particular value for a a given attribute.

Context:
This pattern is used in case two classes have a similar but not completely overlapping extension and the intent of the first class is expressed by a particular attribute value in the target class.
This pattern can be applied to any domain.

Solution:
Solution description:
This pattern establishes a correspondence between a class/attribute/attribute value combination in one ontology and a class in another.
Two classes are related with the scope of one class restricted using an attribute value condition.

Syntax:
```
<ClassByAttributeValue
  rdf:about="ClassByAttributeValueCorrespondence">
 <entity1>
  <Class rdf:about="ClassWithCondition">
   <condition>
    <AttributeCondition>
     <restriction>
      <Restriction
       rdf:about="RestrictionInstance">
       <comparator rdf:datatype="&xsd;string">
        any xsd comparator</omwg:comparator>
       <value rdf:datatype="&xsd;string">
         any value</omwg:value>
       <onProperty rdf:resource="&omwg;nil"/>
      </Restriction>
     </restriction>
    </AttributeCondition>
   </condition>
  </Class>
 </entity1>
 <entity2>
  <Class rdf:about="TargetClass">
 </entity2>
 <measure rdf:datatype='&xsd;float'>1.</measure>
 <relation>equivalence</relation>
</ClassByAttributeValue>
```

Example:
A *BordeauxWine* in the Wine Ontology corresponds to a *Vin* (wine) having for *terroir* (origin) the *Bordeaux* region.
Corresponds to the example given in textitSection 1.2.

Related Patterns: Equivalent Class Correspondence, Subclass Correspondence, Class By Attribute Type Correspondence,
Class By Attribute Occurrence Correspondence

Degenerated Pattern:
Class By Attribute Correspondence

Table 1: The Class By Attribute Correspondence Pattern

a pattern is thus obtained by taking its parent in the class hierarchy.

The patterns library extends the Alignment Ontology under the `Cell` class. Properties corresponding to the pattern template elements are modeled as OWL annotation properties: *patternAlias, patternProblem, patternContext, patternKnownUses, patternResultingContext* and *patternSolutionDescription*. An *rdf:comment* can be added to give a rough description of the pattern. A pattern in the library is modeled as a class with restrictions on its possible instantiations and an example instance is given for each pattern. The deepest one goes into the pattern hierarchy, the more specific are the patterns, making retrieval of patterns easier as suggested in [17]. Patterns at the root of the hierarchy are classified on the type of the entities they are to be applied (Attribute, Class and Relation). *Equivalent* correspondence patterns model the most frequent kind of correspondence usually used to solve a terminological mismatch[6]. *Subsumption* correspondence patterns also model very common correspondence arising when relating entities, they typically solve a granularity mismatch[7]. These two patterns are in the scope of most matching discovery algorithms. They often need to be refined in order to obtain correspondences reflecting more precisely the relation standing between the related entities. Restriction and transformation patterns can be used to refine these correspondences. Union and intersection patterns solve granularity mismatches. Patterns relating heterogeneous entities are used when the structural choices made by the ontology engineer differ to model a similar domain. For example, a relation in one ontology is represented as a class in the other ontology. This patterns are solving structural mismatches.

The library is developped as an extension of the Alignment Ontology using the ontology editor Protege[8], and is available on the web at [18].

Besides patterns themselves, the library provides a set of generic ontological entities used to compose patterns. These entities serve as placeholders to be filled with concrete entity names when using the pattern to build a concrete correspondence. These generic entities provide a reference structure for modeling concrete entities in the same way patterns provide a reference structure for modeling concrete correspondences. For example, the `ClassConditionInstance` models a class with a condition restricting its scope.

A pattern library is said to be complete if all possible instances of a design problem are covered by a pattern in the library [1] . In the case of ontology alignment this means that each possible alignment can be created by instantiating a (number of) pattern(s) and every ontology mismatch can be solved by using one or a combination of patterns. As pointed out in the literature, it is not feasible in general to cover all the possible designs, and moreover impossible to guarantee that no new design problem will arise that is not covered by the pattern library. In our special case of ontology alignment, we have further restrictions on the kind of patterns that can be represented as we are limited by the expressiveness of the alignment representation formalism. We can therefore only define completeness of the library with respect to the expressiveness of this formalism. Also, it is not possible at this stage to say wether the elementary patterns defined in the library will be sufficient to construct complex patterns solving complex alignment problems. Practical extensions of the library for domain specific alignments will help validating and extending this set of elementary patterns.

We ground correspondences using transformations of the Alignment Ontology instances into the target formalism. The Alignment API [9] and the Mapping API [19] propose groundings into various semantic-web languages.

- A grounding to OWL [2] provide the possibility to interpret alignments as OWL ontologies, and thus to execute them using an OWL reasoner. The counterpart is that this ontology language is not expressive enough to model rules and functions necessary to build complex correspondences.

- A grounding to WSML in its rule variant [6] is also provided, allowing to define alignments for mediation between semantic web services descriptions.

- A SKOS [15] grounding using `skos:narrower` and `skos:braoder` to relate entities from thesauri.

- We are currently working on a SPARQL extension [16] able to represent ontology alignments for instance transformation.

4. RELATED WORK

Design patterns were first mentioned in architecture by Alexander in 1977 [1]. 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 [11]. Recent works [21, 3, 12] propose the use of patterns for ontology engineering.

Organization of software patterns was studied in [17]. The ontological organization of patterns provide better retrieval functionalities, particularly fitting in the semantic web environment as we will see later Section 3.

In [7], Clark *et al.* propose the use of Knowledge Patterns for engineering knowledge bases, using morphisms to instantiate the patterns. They particularly distinguished between the pattern, and implementation in a particular programming language. Staab *et al.* [21] propose the use of Semantic Patterns to engineer ontologies while remaining independent from the underlying ontology language. This concept is similar to the separation between correspondence patterns and grounded correspondences we introduced Section 1.1. Gangemi [12] proposes the use of patterns based on experience. He also introduces a pattern template and a library of common patterns for ontology engineering.

[6]Terminological mismatches involve all mismatches due to differences of naming between ontological entities.

[7]Granularity mismatch happen when a part of the domain is more refine in one ontology than in the the other

[8]http://protege.stanford.edu

5. CONCLUSION

We introduced in this paper ontology correspondences patterns as helper to model ontology alignments. We defined a template based on the literature on design patterns and gave a library of elementary correspondence patterns. Patterns instances are described using the Alignment Ontology, a formalism to represent ontology alignments abstract from the underlying knowledge representation formalism. In order to be executed, pattern instances are grounded to the desired formalism. We are constantly refining the library given experience in research projects. There are two application domains where specific patterns could enrich the library: relational database to ontology, and ontology versionning. Relating databases to ontologies becomes an important topic as a considerably large amount of data is already available in relational databases. Publishing this data on the semantic web ask to either convert it to RDF stores or to maintain it in the relational database but provide a mapping to an ontology. This second solution is often preferd as it is more flexible. Providing specific database to ontology patterns will help the task of the specialist designing these mappings. Another application of ontology alignment consists in maintaining consistent versions of evolving ontologies. Changes in one version of an ontology are forwarded to its previous versions via the use of correspondences. Providing specific patterns for change management could help the ontology engineer realizing the alignment between these different versions. We currently investigate on these two topics as possible extensions of the pattern library.

6. ACKNOWLEDGEMENTS

The authors would like to thanks Jos de Bruijn for useful discussions about correspondence patterns.

7. REFERENCES

[1] Christopher Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*, volume 2 of *Center for Environmental Structure Series*. Oxford University Press, New York, New York, USA, 1977.

[2] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL web ontology language reference. Technical report, W3C, 2004. W3C Recommendation 10 February 2004.

[3] Eva Blomqvist and Kurt Sandkuhl. Patterns in ontology engineering: Classification of ontology patterns. In *ICEIS (3)*, pages 413–416, 2005.

[4] A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1:153–184, 2003.

[5] Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. Cowl: Contextualizing ontologies. In *Second International Semantic Web Conference (ISWC-2003), LNCS vol. 2870, pp. 164-179, Springer Verlag, 2003.*, 2003.

[6] Jos de Bruijn, Holger Lausen, Axel Polleres, and Dieter Fensel. The web service modeling language: An overview. In *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, number 4011 in Lecture Notes in Computer Science, pages 590–604, Budva, Montenegro, June 2006. Springer-Verlag.

[7] Peter Clark, John Thompson, and Bruce Porter. Knowledge patterns. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, pages 591–600, San Francisco, 2000. Morgan Kaufmann.

[8] Jos de Bruijn, Douglas Foxvog, and Kerstin Zimmermann. Ontology mediation patterns library v1. Deliverable D4.3.1, SEKT project (IST-2003-506826), 2004.

[9] Jérôme Euzenat. An API for ontology alignment. In *Proc. 3rd international semantic web conference, Hiroshima (JP)*, pages 698–712, 2004.

[10] Jérôme Euzenat, François Scharffe, and Antoine Zimmerman. D2.2.10: Expressive alignment language and implementation. Project deliverable 2.2.10, Knowledge Web NoE (FP6-507482), 2007.

[11] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Pub., 1995.

[12] Aldo Gangemi. Ontology design patterns for semantic web content. In *International Semantic Web Conference*, pages 262–276, 2005.

[13] Michael Klein. Combining and relating ontologies: an analysis of problems and solutions. In *Workshop on Ontologies and Information Sharing*, 2001.

[14] Alexander Mädche, Boris Motik, Nuno Silva, and Raphael Volz. MAFRA – a mapping framework for distributed ontologies. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 235–250, 2002.

[15] Alistair Miles and Ban Brickley. Skos core vocabulary. Technical report, World Wide Web Consortium (W3C), 2005.

[16] Axel Polleres, François Scharffe, and Roman Schindlauer. Sparql++ for mapping between rdf vocabularies. In *ODBASE, On the Move Federated Conferences (to appear)*, 2007.

[17] J.-M. Rosengard and M. F. Ursu. Ontological representations of software patterns. *Lecture Notes in Computer Science (Proceedings of the of KES'04)*, 3215, 2004.

[18] François Scharffe. Omwg d7: Ontology mapping language. `http://www.omwg.org/TR/d7/`, 2007.

[19] François Scharffe and Jos de Bruijn. A language to specify mappings between ontologies. In *Proc. of the Internet Based Systems IEEE Conference (SITIS05)*, 2005.

[20] François Scharffe, Jos de Bruijn, and Douglas Foxvog. Ontology mediation patterns library v2. Deliverable D4.3.2, SEKT project (IST-2003-506826), 2005.

[21] Steffen Staab, Michael Erdmann, and Alexander Maedche. Engineering ontologies using semantic patterns. In A. Preece & D. O'Leary, editor, *Proceedings of the IJCAI-01 Workshop on E-Business & the Intelligent Web, Seattle, WA, USA, August 5, 2001*, 2001.