Pattern Definitions and Semantically Annotated Instances

Ivan Perez[1] and Oscar Corcho[2]

[1] IMDEA Software*. Universidad Politécnica de Madrid, Spain.
ivan.perez@imdea.org
[2] Ontology Engineering Group. Departamento de Inteligencia Artificial.
Universidad Politécnica de Madrid, Spain.
ocorcho@fi.upm.es

Abstract Ontology design patterns are normally instantiated by replicating and adapting the pattern concepts and roles. The relation between pattern definitions and their instantiations is documented in natural language. The use of parametric ontologies or pattern-reuse modifications to OWL-DL has been suggested before, but so far only practical aspects have been analysed, leaving the formal semantics of these extensions as future work. In this work we present formal definitions for ontology pattern and pattern instantiations, together with the semantics of instantiation. We propose the use of semantic annotations to describe and generate OWL pattern instantiations, without the need for explicit ontology replication, and provide tools to support this process.

1 Introduction

Ontology patterns represent knowledge that is subject to appear frequently in different ontologies, normally with different names. Relations such as "being part of something", "participating in an event" and "n-ary relations" are examples of such frequent type of knowledge.

Pattern instantiation is normally done in two different ways: (1) importing the pattern axioms and establishing mappings (equivalences) with existing elements in our ontology, or (2) replicating the pattern by changing the names of its elements. The former leads to representation and reasoning problems in case we use the pattern twice in the same ontology, as described in section 2. The latter is error prone and loses the relation between the instantiations and the pattern, which is in some cases only described with comments in natural language.

Ad-hoc pattern definition and instantiation languages have been proposed for this purpose. However, we propose using the ontology language expressiveness, by means of semantic annotations, to support this pattern instantiation process.

This paper is organised as follows. In section 2 we introduce some of the problems found when trying to reuse ontology design patterns. In section 3

* The work of IMDEA Software on this paper has been partially funded by the Spanish Ministry of Industry, Tourism and Trade under the grant FIT-340503-2007-2, as part of the Morfeo EzWeb strategic singular project.

we define formally the concepts of pattern and pattern instance. In section 4 we describe our tools for supporting pattern reuse. Finally, section 5 presents conclusions, related work and further research topics in this area.
Notation. We use the notation Concept for concepts or classes, property for properties or roles, and *individual* for individuals. The notation used for DL formulas, unless stated otherwise, follows the one presented in [1].

2 Problem description

Let us start with an example where we want to represent cities as part of provinces, which we will use as an ontological pattern. We use the concepts City and Province, the property isPartOf, and the following axioms[3]:

$$\begin{aligned} \top &\sqsubseteq \forall \mathsf{isPartOf}^{-1}.\textsc{City} \\ \top &\sqsubseteq \forall \mathsf{isPartOf}.\textsc{Province} \\ \textsc{City} &\sqsubseteq = 1\ \mathsf{isPartOf}.\top \end{aligned}$$

If we add the concepts GermanCity and GermanProvince, we can reuse the above axiom to relate them. With the following axioms, we can use the relation isPartOf also for German cities and provinces:

$$\begin{aligned} \textsc{GermanCity} &\sqsubseteq \textsc{City} \\ \textsc{GermanProvince} &\sqsubseteq \textsc{Province} \end{aligned}$$

These axioms do not guarantee that a GermanCity belongs to (isPartOf) a GermanProvince. This situation is more clear if we add new subconcepts of City and Province, for instance, FrenchCity and FrenchProvince:

$$\begin{aligned} \textsc{FrenchCity} &\sqsubseteq \textsc{City} \\ \textsc{FrenchProvince} &\sqsubseteq \textsc{Province} \end{aligned}$$

According to our axioms, it could be the case that a FrenchCity is part of a GermanProvince. To solve this, we may add the following axioms:

$$\begin{aligned} \textsc{GermanCity} &\equiv \forall \mathsf{isPartOf}.\textsc{GermanProvince} \\ \textsc{FrenchCity} &\equiv \forall \mathsf{isPartOf}.\textsc{FrenchProvince} \end{aligned}$$

This solution is correct for our case. Note, however, that we have proposed an ad-hoc solution to a simple problem of multiple instantiation of one pattern. In larger ontologies, it may be easier to simply duplicate the original pattern using different names for each element in the pattern:

[3] We use this adhoc "pattern" to simplify our explanation, since we have not found any well-known ontology pattern that contained these axioms and could be used instead. Potential candidate patterns like Componency [12], Collection [4], Collection Entity [11] or Classification [10] lacked some of the concepts or axioms.

$$\begin{aligned} \top &\sqsubseteq \forall \mathsf{gIsPartOf}^{-1}.\textsc{GermanCity} \\ \top &\sqsubseteq \forall \mathsf{gIsPartOf}.\textsc{GermanProvince} \\ \textsc{GermanCity} &\sqsubseteq =1\ \mathsf{gIsPartOf}.\textsc{GermanProvince} \end{aligned}$$

with a similar description for the concepts FrenchCity and FrenchProvince, and the role fIsPartOf. To document the relation between the pattern and its instances, we may keep the subclass relations above and add the following axioms:

$$\begin{aligned} \mathsf{gIsPartOf} &\sqsubseteq \mathsf{isPartOf} \\ \mathsf{fIsPartOf} &\sqsubseteq \mathsf{isPartOf} \end{aligned}$$

but this adds one new axiom for each element that is part of an instance of a pattern, and that relation is often just documented in natural language (as an `rdfs:comment` associated to the role).

3 Pattern definitions and instantiations

Two key elements are needed in ontologies to support patterns: a means to define the patterns and a way to use them. Before going into details, we introduce a few elementary definitions that will be used along the paper. We use C for a set of concept names, R for a set of role names, and I for individual names. We consider a DL, with a set of symbols S, as a language over the union of all four sets C, R, I and S. DL sentences are formulas, and an ontology is a finite set of formulas of a particular DL. We consider C^+, R^+, I^+ and S^+ pairwise disjoint[4]. Finally, given a set W, a list of symbols $w_1, \ldots, w_n \in W$ all different, and a second list of symbols $w'_1, \ldots, w'_n \in W$, we define the substitution function from $w_1, \ldots, w_n \in W$ to $w'_1, \ldots, w'_n$ respectively, as the function $f \subseteq W \times W$ such that:

$$f(x) = \begin{cases} w'_i & \text{if there is an } i \in \{1, \ldots, n\} \text{ such that } w_i = x \\ x & \text{otherwise} \end{cases}$$

We refer to that substitution function as the mapping $[w_1 \mapsto w'_1, \ldots, w_n \mapsto w'_n]$ or $[f]$. Given a substitution function f, and given a string w of elements in W of length n (that is, $w : [n] \to W$)[5], we define the substitution of w under the total substitution function $f : W \to W$, and we represent it as $w[f]$, as the string $w' : [n] \to W$ such that $\forall i \in [n]\ w'(i) = f(w(i))$.

3.1 Pattern definitions

Ontology patterns are defined as DL models or UML diagrams, plus descriptions in natural language. If we use DL, we have no standard way of establishing

[4] We use the notation X^+ for the Kleene plus, that is, the Kleene closure of the set X without the empty string. More details can be found in [3].

[5] We follow the notation for strings and alphabets in [3]. $[n]$, where n is a natural number, represents the subset of natural numbers from 1 to n.

which parts of the model are meant to be substituted. In case we use UML, the semantics in ontologies when the pattern is instantiated is not clear.

In our proposal, we also use DL models to define patterns. However, we add an interface to the pattern, that is, a definition of which parts are instantiable. This allows us to establish if an element in a pattern is not meant to be substituted or instantiated. This can happen because a name is just presented to simplify the axioms, or if the element in question represents more general knowledge.

As an example, we will use the model in section 2, with concepts City and Province, the property isPartOf, and the following axioms:

$$\begin{aligned} \top &\sqsubseteq \forall \mathsf{isPartOf}^{-1}.\textsc{City} \\ \top &\sqsubseteq \forall \mathsf{isPartOf}.\textsc{Province} \\ \textsc{City} &\sqsubseteq = 1\ \mathsf{isPartOf}.\top \end{aligned}$$

This model and the interface $\{\textsc{City}, \textsc{Province}, \mathsf{isPartOf}\}$ is a pattern definition.

Definition 1. (Ontological knowledge pattern)
Given an ontology O, and given the sets $C' \subseteq C$, $R' \subseteq R$ and $I' \subseteq I$. The tuple $\langle O, C', R', I' \rangle$ is an ontological knowledge pattern.

Unless stated otherwise, we will normally refer to ontology patterns as pattern definitions or simply patterns. If $\langle O, C', R', I' \rangle$ is a pattern, we will refer to C', R' and I' as the interface of the pattern.

3.2 Pattern instantiations

A model is an instantiation of a pattern with a parameter/value assignation if the model has exactly the same axioms as the pattern, where the parameters have been substituted with their corresponding values. Note that the values must be of the same kind, that is, we can substitute or instantiate a concept with a concept, a role with a role and an individual with an individual.

Given the pattern definition above, we can instantiate it with the assignations FrenchCity to City and FrenchProvince to Province, having as a result the following model:

$$\begin{aligned} \top &\sqsubseteq \forall \mathsf{isPartOf}^{-1}.\textsc{FrenchCity} \\ \top &\sqsubseteq \forall \mathsf{isPartOf}.\textsc{FrenchProvince} \\ \textsc{FrenchCity} &\sqsubseteq = 1\ \mathsf{isPartOf}.\top \end{aligned}$$

The following is a formal definition of pattern instantiation:

Definition 2. (Pattern instantiation)
Let $\langle O, C', R', I' \rangle$ be a pattern, where O is an ontology over some description logic D. Let $c_1, \ldots, c_m \in C', k_1, \ldots, k_m \in C$ be concepts, $r_1, \ldots, r_n \in R'$, $s_1, \ldots, s_n \in R$ be roles, and $i_1, \ldots, i_p \in I', j_1, \ldots, j_p \in I$ be individuals.

We say that O' is an instance of the pattern $\langle O, C', R', I' \rangle$ with the mappings $c_1 \mapsto k_1, \ldots, c_m \mapsto k_m, r_1 \mapsto s_1, \ldots, r_n \mapsto s_n, i_1 \mapsto i_1, \ldots, i_p \mapsto j_p$ if $O' = \{o[c_1 \mapsto k_1, \ldots, c_m \mapsto k_m, r_1 \mapsto s_1, \ldots, r_n \mapsto s_n, i_1 \mapsto i_1, \ldots, i_p \mapsto j_p] \mid o \in O\}$ and O' is also an ontology over D.

3.3 Rebasing ontologies to avoid name clashing

We now address a practical issue commonly found when *partially* instantiating the same pattern twice as part of the same ontology. Imagine that we instantiate the city pattern for French and German cities, and do not want to name isPartOf differently in each case. The following axioms would be part of the result:

$$\top \sqsubseteq \forall \mathsf{isPartOf}^{-1}.\textsc{GermanCity}$$
$$\top \sqsubseteq \forall \mathsf{isPartOf}^{-1}.\textsc{FrenchCity}$$

which implies that either isPartOf is empty or GermanCity and FrenchCity are related (one is a subclass of the other).

Finding these problems may not be so obvious. Besides, name clashes may occur in non-instantiable elements. The pattern designer may document or avoid these issues, but it is the responsibility of the pattern user to make sure that no inconsistencies are introduced with multiple instantiations of a pattern.

Name clashes are not necessarily a mistake from a formal point of view, so these are only guidelines to pattern design. However, we will help avoid these situations by means of namespace (or URI) translations.

URIs and ontology rebasing. When coded in OWL, ontologies have a base URI and all entities local to it have that URI as a prefix of their complete names. By changing the base URI of an ontology, we are effectively renaming all the local entities at once, thus avoiding all name clashes of non-instanced local names.

In our formal definitions, we consider the unqualified name sets C_u, R_u and I_u, respectively, for concepts, names and individuals. We also consider a set M of namespaces, such that $C = M \times C_u$, $R = M \times R_u$, and $I = M \times I_u$. Like before, C_u, R_u and I_u are pairwise disjoint.

Definition 3. (Ontology rebasing)
Let O be an ontology, and $m, m' \in M$ two namespaces. We define the rebasing of O from m to m', and we represent it as $O^{m \mapsto m'}$, as the ontology $\{o[f] \mid o \in O\}$ where f is the substitution function defined by the relation $\{(x, y) \in C \cup R \cup I \times C \cup R \cup I \mid \exists z \in (C_u \cup R_u \cup I_u)\ x = \langle m, z \rangle \wedge y = \langle m', z \rangle\}$.

The previous definition of pattern instantiation is now extended as follows:

Definition 4. (Pattern instantiation)
We say that an ontology O is an instantiation of the pattern P with the namespace change from m to m' and the mappings $[e_1 \mapsto d_1, \ldots, e_n \mapsto d_n]$ if $O = O'^{m \to m'}$ and O' is an instantiation of the pattern P with the mappings $[e_1 \mapsto d_1, \ldots, e_n \mapsto d_n]$.

For example, assume we use `http://foo/Cities` as the base URI of the cities pattern and `http://foo/Cities#isPartOf` as the qualified name of isPartOf. If we set `http://foo/FCities` and `http://foo/GCities` as the base URIs of the instances for French and German cities respectively, applying the new pattern instance definition with the same mappings as before would give us the following axioms instead, where no name clashes occur:

$$\top \sqsubseteq \forall \texttt{http://foo/GCities\#isPartOf}^{-1}.\texttt{http://foo/GCities\#GermanCity}$$
$$\top \sqsubseteq \forall \texttt{http://foo/FCities\#isPartOf}^{-1}.\texttt{http://foo/FCities\#FrenchCity}$$

4 Tool support

We provide tool support for the processes of pattern definition and instantiation.

Pattern definition. Patterns are defined as ontologies with an interface, hence to define patterns we simply need to support *interface declaration*. For this purpose, we have developed:

- An ontology [7] with the annotation property exportable, which can be set for any class, property or individual, and is `true` if the element is part of the interface (instantiable) or `false` otherwise. Its default value is assumed to be `true`, so that users do not need to annotate all the elements in a pattern.
- A Protégé plugin [6] to assign values for this annotation property.

Pattern instantiation and use. Defining pattern instances means identifying the pattern to be used, establishing a parameter/value map and a URI rebasing. This task is supported with the following elements:

- An ontology [9] to describe instantiations. It contains concepts to represent mappings between entities and URI rebases.
- A Protégé plugin [6] that eases the creation of these instantiation A-Boxes. It also allows the user to apply a particular instantiation.
- An ontology [8] with an annotation property called `isPatternInstance`, that indicates that an ontology is the result of applying a particular pattern instance definition as defined in the first step.

The new Protégé plugin is accessible under the menu Tools, as an import *wizard*. The process is divided in three steps. First, the location of a pattern definition (an ontology) is provided. Second, the plugin shows all the elements in that pattern that are *exportable*. The user can then introduce the main information about the instantiation, that is, the new names for each entity that will be instantiated, and the URI rebase. Third, the plugin allows the user to select a location where the instantiation will be saved. This will create an ontology with all the information of this particular instantiation (the entity mappings and the URI rebase), located in a different file. A screenshot is shown in Fig. 1.

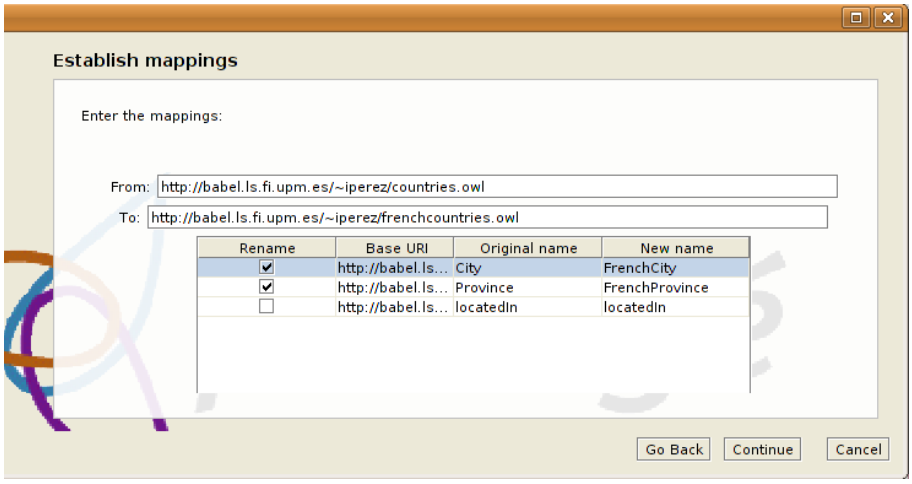

Figure 1. Screenshot of the instantiation wizard plugin running in Protégé 4

Once the process is finished, the plugin loads the pattern, applies the URI rebase and entity mapping, and adds all the axioms into the currently active ontology. It also annotates the ontology with the property `isPatternInstance`, using the base URI of the pattern instance definition as value.

5 Conclusions and future work

In this paper we have shown how patterns can be defined as ontologies with an interface (as if they were parametric ontologies), and how pattern instantiations can be formalised as a substitution of the parameters in a pattern ontology, providing formal definitions for both concepts. We have also analysed some of the problems derived from multiple pattern instantiation, and suggested ontology rebasing as a way to avoid name clashes.

Some notable work in pattern definition is presented in [5], where patterns are presented as part of an ontology modification language (OPPL), not focusing on pattern reuse specifically. There are also some relationships with Package-Based Description Logics [2], where three views to entities are provided: public, protected and private. The authors of Package-Based Description Logics state that this change introduces parameterism in ontologies, although its expressiveness is not explored.

The current OWL syntax and tool support is built completely around annotation properties. Protégé plugins are based on the OWLAPI library, and they are currently in an early beta-testing state of development. In future versions of the syntax, we plan to keep annotation properties to indicate if an element is instantiable in a pattern, and add support for pattern imports to the modules mechanism in OWL. This could be done by extending `owl:imports` with a nested

tag `owl:patterninstance`, which in turn would have `from_uri` and `to_uri` attributes for the rebasing. Also, nested `owl:mapping` elements with attributes `from_name` and `to_name` could be used for the element/value assignations. The meaning of these attributes, their domain and range, would be according to what was established in the Pattern Instance definition ontology, in section 4.

Regarding the expressiveness of the pattern definition language, it currently allows roles, concepts and individuals to be treated as parameters. This may be sufficient in most cases, but it might be interesting to have other parametrisable parts of an ontology. For instance, numbers in cardinality restrictions could also be parameters. Note that the introduction of parametricity at this level would likely make the pattern definition no longer be an ontology.

Even though some content ODPs can easily be represented with our definition of pattern, it is not clear that all ODPs can. Future research should also focus on trying to find Ontology Design patterns that cannot be represented with our definition, in order to identify other elements that can also be parameterised.

References

1. Franz Baader and Werner Nutt. Basic description logics. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *Description Logic Handbook*, pages 43–95. Cambridge University Press, 2003.
2. Jie Bao, Doina Caragea, and Vasant Honavar. On the semantics of linking and importing in modular ontologies. In *International Semantic Web Conference*, pages 72–86, 2006.
3. Jean H. Gallier. *Logic for computer science: foundations of automatic theorem proving*. Harper & Row Publishers, Inc., New York, NY, USA, 1985.
4. Aldo Gangemi. Collection design pattern. http://ontologydesignpatterns.org/wiki/Submissions:Collection.
5. Luigi Iannone, Alan Rector, and Robert Stevens. Embedding knowledge patterns into OWL. In *6th Annual European Semantic Web Conference (ESWC2009)*, pages 218–232, June 2009.
6. Iván Pérez. Ontology engineering protege plugins, 2009. IMDEA Software. http://sharesource.org/project/ontoengineeringprotegepluglins/.
7. Iván Pérez. Pattern definition ontology, 2009. IMDEA Software. http://babel.ls.fi.upm.es/~iperez/pattern-ontologies/patterndefinition.
8. Iván Pérez. Pattern instance ontology, 2009. IMDEA Software. http://babel.ls.fi.upm.es/~iperez/pattern-ontologies/patterninstance.
9. Iván Pérez. Pattern Instance definition ontology, 2009. IMDEA Software. http://babel.ls.fi.upm.es/~iperez/pattern-ontologies/patterninstancedefinition.
10. Valentina Presutti. Classification design pattern. http://ontologydesignpatterns.org/wiki/Submissions:Classification.
11. Valentina Presutti. Collection Entity design pattern. http://ontologydesignpatterns.org/wiki/Submissions:CollectionEntity.
12. Valentina Presutti. Componency design pattern. http://ontologydesignpatterns.org/wiki/Submissions:Componency.