WOP 2012

3rd Workshop on Ontology Patterns

Co-located with ISWC2012
Boston (MA), USA - November 12th 2012

Proceedings

Edited by:
Eva Blomqvist, Human-Centered Systems, Linköping University (SE)
Aldo Gangemi, Semantic Technologies Lab, ISTC-CNR (IT)
Karl Hammar, Information Engineering, Jönköping University (SE)
Mari Carmen Suárez-Figueroa, Ontology Engineering Group (OEG),
Universidad Politécnica de Madrid (UPM) (ES)

Preface
- The 3rd Workshop on Ontology Patterns - WOP 2012

During the past two years, the notion of Ontology Patterns has clearly become more mainstream within the Semantic Web field. We have for instance noted that in last years ISWC one full paper session, entitled Ontologies and Patterns, was devoted almost entirely to patterns. This emerging role of patterns is also evident from the number of paper submissions for this years edition of WOP (see further below), which has broken all previous records.

Since the beginning of the Semantic Web initiative, ontologies have been referred to as the key tool for implementing the Semantic Web vision. However, with the success of Linked Data as a highly data-focused movement, ontologies are now challenged to respond to new needs, such as handling big and noisy data. Ontologies are also often built to reflect the reality in data, rather than the other way around. Ontology Patterns, and their related technologies, hold the potential for bridging the gap between linked data and ontologies, because they are conceived with simplicity, scalability, and modularity, as well as contributions from, and usage by, the masses in mind, without giving up the inheritance from Artificial Intelligence and philosophical ontologies. Hence, we envision Ontology Patterns as a necessary facilitator in the upcoming breakthrough for the Semantic Web on the larger social Web arena.

The aim of WOP is to give researchers and practitioners a forum for sharing their latest findings and emerging issues, as well as building a common language of Ontology Patterns. Furthermore, the WOP community is supported by the ontologydesignpatterns.org initiative, and uses it as its main mean of communication, e.g. for pattern submission, reviewing and discussion outside the workshop schedule. This support gives the workshop a continuity and a platform for discussion that few others have.

This years WOP instance is the third in the series. We received 14 full research paper submissions and 5 additional pattern submissions. The members of the Program Committee acknowledge the very high quality of many of these submissions, as 10 of them were recommended for acceptance. As this year the workshop was selected for only a half-day slot, the chairs then decided to also accept a number of papers for poster presentation. In total 6 papers were accepted for oral presentation (a 43% acceptance rate), and 4 for poster presentation. In the pattern track, 3 out of 5 submissions were accepted for presentation as pattern posters. The pattern submissions consisted of both an extended abstract and an actual pattern uploaded on the ontologydesignpatterns.org portal.

Further information about the Workshop on Ontology Patterns can be found at: http://ontologydesignpatterns.org/wiki/WOP:2012.

Acknowledgments

We appreciate support from the DEON project (funded by STINT) and send special thanks to all members of the steering committee, program committee, authors and local organizers for their efforts and support.

WOP Paper Chairs
Aldo Gangemi and Eva Blomqvist

WOP Pattern Chairs
Mari Carmen Suárez-Figueroa and Karl Hammar

October 2012

Table of Contents

Research Papers

Specifying Ontology Design Patterns with an Ontology Repository 1
Michael Gruninger and Megan Katsumi

Ontology Design Patterns in Use - Lessons Learnt from an Ontology Engineering Case 13
Karl Hammar

Ontology Design Pattern Language Expressivity Requirements 25
Matthew Horridge, Mikel Egaña Aranguren, Jonathan Mortensen, Mark Musen and Natalya Noy

Modest Use of Ontology Design Patterns in a Repository of Biomedical Ontologies 37
Jonathan M. Mortensen, Matthew Horridge, Mark A. Musen and Natalya F. Noy

A Pattern For Interrelated Numerical Properties 49
Jesper Zedlitz, Hagen Peters and Norbert Luttenberger

Linguistic Patterns for Information Extraction in OntoCmaps 61
Amal Zouaq, Dragan Gasevic and Marek Hatala

Poster Papers

Applications of Ontology Design Patterns in the Transformation of Multimedia Repositories 73
Agnieszka Lawrynowicz and Raul Palma

SPARQL-DL queries for Antipattern Detection 85
Catherine Roussey, Oscar Corcho, Ondrej Zamazal, Francois Scharffe and Stephan Bernard

Relational Database to RDF Mapping Patterns 97
Juan F. Sequeda, Freddy Priyatna and Boris Villazon-Terrazas

Building Ontologies by using Re-engineering Patterns and R2RML Mappings 109
Boris Villazón-Terrazas and Freddy Priyatna

Pattern Abstracts

The Template Instance Pattern 121
Csongor I Nyulas, Tania Tudorache and Samson Tu

Conformance to standards 125
Monika Solanki and Craig Chapman

Reactive Processes 129
Monika Solanki and Craig Chapman

Specifying Ontology Design Patterns with an Ontology Repository

Michael Grüninger and Megan Katsumi

Department of Mechanical and Industrial Engineering, University of Toronto
Toronto, Ontario, Canada M5S 3G8
gruninger@mie.utoronto.ca, katsumi@mie.utoronto.ca

Abstract. Within the Common Logic Ontology Repository (COLORE), the notion of reducibility among ontologies has been used to characterize relationships among ontologies. This paper uses techniques such as relative interpretation to show how one set of ontologies within the repository can be reused to characterize the models of other ontologies that are used in a wide variety of domains. A central theme of the paper is that ontology design patterns can be formalized as core ontologies within the ontology repository.

1 Introduction

The COLORE (Common Logic Ontology Repository) project[1] is building an open repository of first-order ontologies that serve as a testbed for ontology evaluation and integration techniques, and that can support the design, evaluation, and application of ontologies in first-order logic. The logical relationships among the set of first-order ontologies in the repository can also be used as basis for the verification of an ontology with respect to its intended models as well as decomposition of ontologies into modules.

We will show how COLORE follows the vision of ontology design patterns as proposed in [6] and [5]. Different notions of ontology design patterns have been used, ranging from syntactic criteria to structural properties of ontologies. As a result, several methodological questions remain challenges – How can we evaluate ontology design patterns and their application? How are design patterns reused? Within COLORE, design patterns are formalized as core ontologies within the repository. Patterns are reused via the metatheoretic relationships of relative interpretation and definable equivalence. In this sense, the ontology design patterns within COLORE are semantic (model-theoretic) rather than syntactic. On the other hand, the approach described in this paper can also be used to generate axioms for new ontologies, in which case we can consider core ontologies to serve as syntactic templates for axioms.

After an informal discussion of ontology design patterns in the context of COLORE, we give an overview of the relationships between ontologies within COLORE. The notions of relative interpretation, definable equivalence, and reduction play a key role in formalizing the reuse of ontologies. In particular, these notions give us techniques for evaluating ontology design patterns and proving that a pattern is correctly and completely exemplified by a set of ontologies. We will illustrate this approach using sets of ontologies within COLORE.

[1] http://code.google.com/p/colore/source/browse/trunk/

2 COLORE and Ontology Design Patterns

In general, Ontology Design Patterns (OPs) are meant to serve as reusable solutions for various aspects of ontology design [6], and the structure of the ontologies in COLORE and the relationships defined between them can provide similar support. COLORE provides a means of sharing content ontology design patterns (CPs) while providing solutions that address specific instances of some of the modelling problems that other OPs are designed to solve.

Of the six families of OPs recognized in [6], the Structural, Correspondence, and Content families of OPs have strong parallels in COLORE:

Structural OPs include what are referred to as Logical and Architectural OPs. Architectural OPs represent possible structures for an ontology being designed. These structures are meant to assist with design choices when computational complexity is a concern, and also to serve as reference material to guide designers in creating their own structures. In particular, *external* Architectural OPs provide patterns for ontology modularization, ("meta-level constructs"). Examples of these *external* Architectural OPs can be found in COLORE as each ontology is stored in modules[9] that are connected to form the ontology using the imports relation.

Correspondence OPs include what are referred to as Reengineering and Mapping OPs. Mapping OPs provide a means to describe the relationship(s) that exist between elements in different ontologies. Similarly, relationships are defined between the terms used in different ontologies in COLORE. In this way the relationships represent specific instances of Mapping OPs. Relationships between ontologies themselves are also described so that users may compare their semantics; these relationships are based on the notion of reducibility discussed in the following section.

Content OPs (CPs) appear to be the most widely used family of OPs. They are typically domain oriented and provide axioms that are intended to be reused as "building blocks" in order to construct an ontology. CPs can also serve other functions in ontology development such as evaluation. Although they are not necessarily domain-oriented, we view the *core theories* of COLORE to be examples of useful CPs, as all ontologies in COLORE are reducible to sets of these ontologies. Using the notion of *intended models*, the core theories in COLORE can also be used for ontology verification ([11],[8]).

3 Relationships between Ontologies in COLORE

The sets of ontologies within COLORE are organized based on the notion of the reduction of one ontology to a set of ontologies. In this section, we review the background for understanding reduction and the role it plays in organizing ontologies within the repository.

3.1 Relative Interpretation

The notion of interpretability between theories[2] is widely used within mathematical logic and applications of ontologies for semantic integration [14]. We will adopt the definition of relative interpretation from [4], in which the mapping π is an interpretation of a theory T_1 with language. L_1 into a theory T_2 with language L_2 iff it preserves the theorems of T_1.

Definition 1. *An interpretation π of a theory T_1 into a theory T_2 is faithful iff*

$$T_1 \not\models \sigma \Rightarrow T_2 \not\models \pi(\sigma)$$

for any sentence $\sigma \in \mathcal{L}(T_1)$.

Thus, the mapping π is a faithful interpretation of T_1 if it preserves satisfiability with respect to T_1. We will also refer to this by saying that T_1 is faithfully interpretable in T_2.

Definable equivalence is a generalization of the notion of logical equivalence to theories that do not have the same signature.

Definition 2. *Two theories T_1 and T_2 are definably equivalent iff T_1 is faithfully interpretable in T_2 and T_2 is faithfully interpretable in T_1.*

For example, the theory of timepoints is definably equivalent to the theory of linear orderings. On the other hand, although the theory of partial orderings is faithfully interpretable in the theory of timepoints, these two theories are not definably equivalent, since the theory of timepoints is not interpretable in the theory of partial orderings.

Definition 3. *Let T_0 be a theory with signature $\Sigma(T_0)$ and let T_1 be a theory with signature $\Sigma(T_1)$ such that $\Sigma(T_0) \cap \Sigma(T_1) = \emptyset$.*
Translation definitions for T_0 into T_1 are sentences in $\Sigma(T_0) \cup \Sigma(T_1)$ of the form

$$\forall \overline{x}\ p_i(\overline{x}) \equiv \Phi(\overline{x})$$

where $p_i(\overline{x})$ is a relation symbol in $\Sigma(T_0)$ and $\Phi(\overline{x})$ is a formula in $\mathcal{L}(T_1)$.

Translation definitions can be considered to be an axiomatization of the interpretation of T_0 into T_1. As noted in the previous section, the use of translation definitions in COLORE is similar to Mapping OPs, insofar as they specify relationships between terms used in different ontologies in order to compare their semantics.

---

[2] In this paper, we consider an ontology to be a set of first-order sentences (axioms) that characterize a first-order theory, which is the closure of the ontology's axioms under logical entailment.

The non-logical lexicon (signature) of a first-order theory T, denoted by $\lambda(T)$, is the set of all constant symbols, function symbols, and relation symbols that are used in T.

The language of T, denoted by $\mathcal{L}(T)$, is the set of all first-order formulas that only use the non-logical symbols in the signature $\lambda(T)$.

3.2 Hierarchies

If an ontology is characterized by its set of ontological commitments, then such commitments will be formalized by sets of axioms. Moreover, in order for the commitments to be comparable, their axiomatizations need to be expressed in the same language. Using these intuitions, we can define an ordering over a set of theories:

Definition 4. *A hierarchy* $\mathbb{H} = \langle \mathcal{H}, < \rangle$ *is a partially ordered, finite set of ontologies* $\mathcal{H} = T_1, ..., T_n$ *such that*

1. $\mathcal{L}(T_i) = \mathcal{L}(T_j)$*, for all* i, j*;*
2. $T_1 \leq T_2$ *iff*

$$T_1 \models \sigma \Rightarrow T_2 \models \sigma$$

for any $\sigma \in \mathcal{L}(T_1)$.

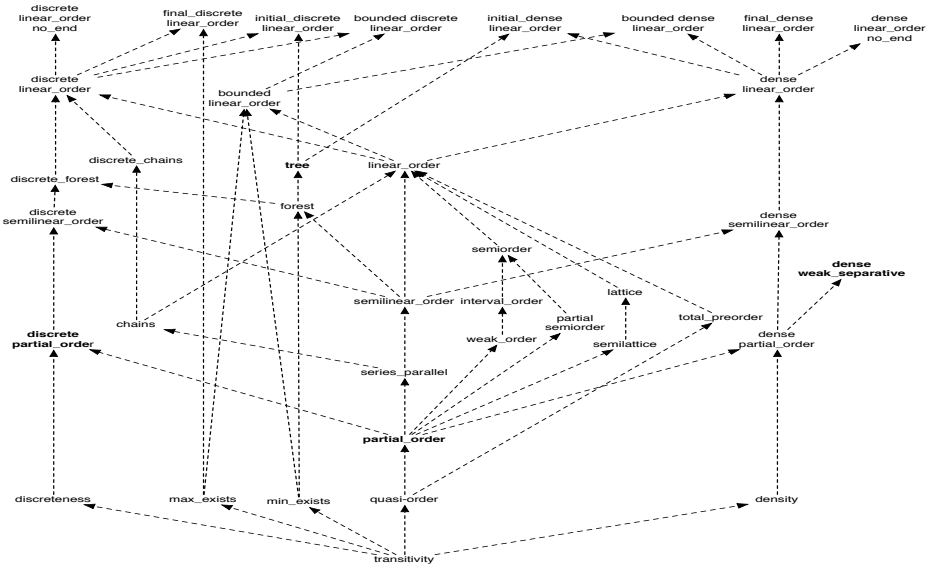


Fig. 1. Ontologies in $\mathbb{H}^{ordering}$: the core hierarchy of orderings. Dashed lines denote nonconservative extension. Theories in bold are ones which are used in this paper.

The theories within two hierarchies in COLORE are shown in Figures 1 and 2. The Ordering Hierarchy[3] contains ontologies that axiomatize different classes of orderings, such as partial orderings, linear orderings, trees, and lattices.

The Mereology Hierarchy[4] contains ontologies that axiomatize different intuitions related to the concept of parthood (see [15] for a full discussion of these ontologies).

[3] http://code.google.com/p/colore/source/browse/trunk/ontologies/core/ordering

[4] http://code.google.com/p/colore/source/browse/trunk/ontologies/core/mereology

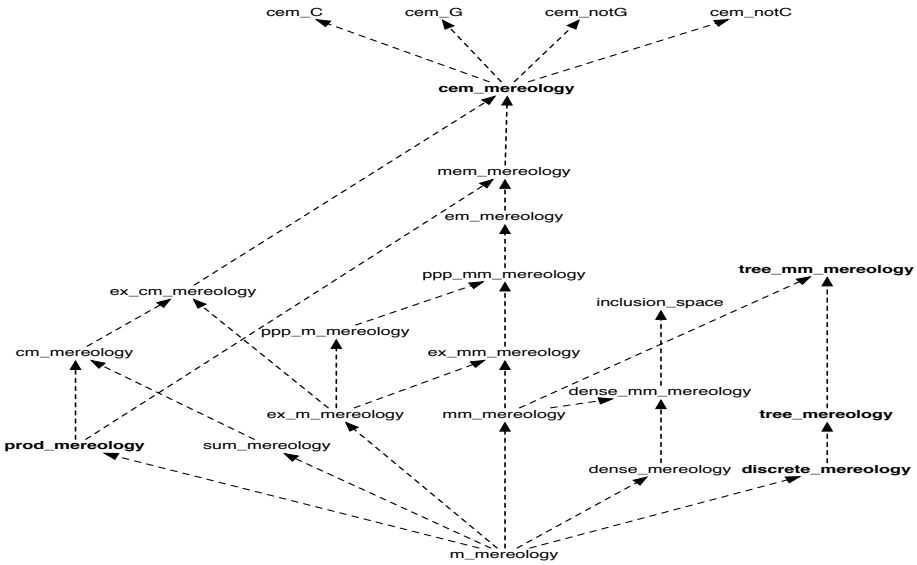


Fig. 2. Ontologies in $\mathbb{H}^{mereology}$: the hierarchy of mereologies. Dashed lines denote nonconservative extension. Theories in bold are ones which are used in this paper.

Note that all extensions of ontologies in the same hierarchy are nonconservative. An ontology T is a root ontology iff it is not the extension of any other ontology in the same hierarchy. Within the $\mathbb{H}^{ordering}$ Hierarchy, the root ontology is the axiomatization of a transitive relation. Within the $\mathbb{H}^{mereology}$ Hierarchy, the root ontology is the axiomatization of a basic mereology (in which the parthood relation is transitive, reflexive, and antisymmetric). This ontology is definably equivalent to the theory $T_{partial\_order}$ within the $\mathbb{H}^{ordering}$ Hierarchy.

3.3 Reducibility

Definable equivalence is a relationship between two ontologies; we can generalize this to a relationship among sets of ontologies. The basis for this approach is the model-theoretic notion of reducibility introduced in [8].

Definition 5. *A ontology T is reducible to a set of ontologies $T_1, ..., T_n$ iff*

1. *T faithfully interprets each T_i, and*
2. *$T_1 \cup ... \cup T_n$ faithfully interprets T.*

We will also refer to the set of ontologies $T_1, ..., T_n$ in the definition as the reduction of T in the repository.

It is easy to see that two definably equivalent ontologies are reducible to each other. For example, within COLORE, the ontology $T_{mereology}$ is reducible to the ontology $T_{linear\_ordering}$ and vice versa.

The following result from [9] characterizes the relationship between reducibility and definable equivalence, and it will be used in this paper to prove results about reducibility:

Theorem 1. *Let $T_1, ..., T_n$ be a set of ontologies such that $\Sigma(T_i) \cap \Sigma(T_j) = \emptyset$ for all $1 \leq i, j \leq n, i \neq j$.*

A ontology T is reducible to $T_1, ..., T_n$ iff T is definably equivalent to $T_1 \cup ... \cup T_n$.

Section 4 will present the reductions of several different ontologies, and discuss their relationship to design patterns.

3.4 Core and Complex Hierarchies

The notion of the reducibility of ontologies can be used to specify an ordering on the set of hierarchies.

Definition 6. *Let $\mathbb{H}_1, ..., \mathbb{H}_n$ be a finite set of hierarchies.*

A repository $\mathbb{R} = \langle \mathcal{R}, \sqsubseteq \rangle$ is a partially ordered set where

- *$\mathcal{R} = \{\mathbb{H}_1, ..., \mathbb{H}_n\}$;*
- *$\mathbb{H}_i \sqsubseteq \mathbb{H}_j$ iff each root ontology in $\mathbb{H}_j$ has a reduction that contains a ontology T in $\mathbb{H}_i$.*

For example, we can show that $\mathbb{H}_{ordering} \sqsubseteq \mathbb{H}_{mereology}$, since the root ontology in $\mathbb{H}_{mereology}$ is definably equivalent to the ontology $T_{partial\_ordering}$ in $\mathbb{H}_{ordering}$. On the other hand, $\mathbb{H}_{mereology} \not\sqsubseteq \mathbb{H}_{ordering}$, since the root ontology for $\mathbb{H}_{ordering}$ (which is $T_{transitive}$) is not reducible to any ontology in $\mathbb{H}_{mereology}$.

Since we are dealing with repositories that contain a finite set of hierarchies, we are guaranteed that the partial ordering $\sqsubseteq$ has minimal elements.

Definition 7. *A hierarchy $\mathbb{C} = \langle \mathcal{C}, \leq \rangle$ is a core hierarchy iff it is a minimal hierarchy in the repository $\mathbb{R} = \langle \mathcal{R}, \sqsubseteq \rangle$.*

An ontology T is a core ontology theory iff it is in a core hierarchy.

A complex hierarchy $\mathbb{H} = \langle \mathcal{H}, \leq \rangle$ is a hierarchy which is not minimal in the repository $\langle \mathcal{R}, \sqsubseteq \rangle$.

An ontology T is a complex ontology iff it is in a complex hierarchy.

Through the notion of reducibility, we can see that core ontologies play the role of building blocks for all other ontologies within the repository. A complex ontology is either constructed from a set of core ontologies or it is an ontology that imposes additional ontological commitments on a core ontology (e.g. the root theory of theory of the $\mathbb{H}^{mereology}$ Hierarchy imposes additional ontological commitments that make the parthood relation reflexive and antisymmetric). If the repository contains multiple equivalent core hierarchies, then the reduction will contain multiple definably equivalent core ontologies, and hence there might exist multiple reductions that contain different sets of core ontologies.

Within COLORE, the notion of a core ontology is therefore based on the logical notion of reducibility, rather than on the distinction between generic vs domain ontologies, as in [13].

4 Hierarchies as Design Patterns

Core ontologies within the repository can be definably equivalent to multiple ontologies in other hierarchies. In this sense, they play the role of design patterns that are reused to verify other ontologies; that is, they can be used to prove that the intended models of an ontology are isomorphic to the models of the axiomatization of the ontology. In this section, we consider in detail one set of core ontologies and show how its relationships to a surprising variety of other ontologies from remarkably different domains.

4.1 Subposet Hierarchy

Each ontology in the Subposet Hierarchy[5] is an extension of an ontology from the Mereology Hierarchy and an ontology from the Ordering Hierarchy.

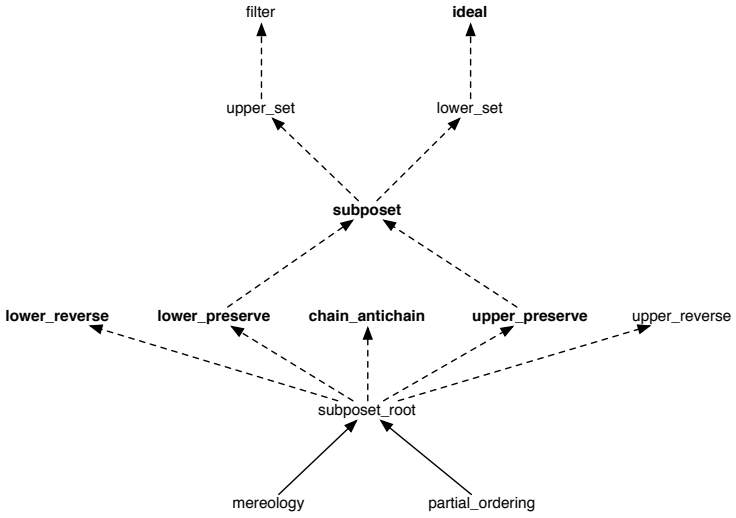


Fig. 3. Ontologies in $\mathbb{H}^{subposet}$: the hierarchy of theories of relationships between partially ordered sets. Dashed lines denote nonconservative extension and solid lines denote conservative extension. Ontologies in bold are ones which are used in this paper.

The ontologies shown in Figure 3 form the basis for the $\mathbb{H}^{subposet}$ Hierarchy. The root ontology $T_{subposet\_root}$ is the union of $T_{m\_mereology}$ and $T_{partial\_ordering}$, and is a conservative extension of each of these ontologies. Thus, each model of $T_{subposet\_root}$ (and hence each model of any ontology in the hierarchy) is the amalgamation of a mereology substructure and a partial ordering substructure.

The ontologies shown in Figure 3 contain additional axioms that constrain how the mereology is related to the partial ordering. In models of $T_{subposet}$, the mereology is a

[5] http://code.google.com/p/colore/source/browse/trunk/ontologies/core/subposet/

subordering of the partial ordering. T_{ideal} strengthens this condition by requiring that the mereology is a subordering of the partial ordering which forms an ideal. In models of $T_{chain\_antichain}$, elements that are ordered by the mereology are not comparable in the partial ordering.

All ontologies within the $\mathbb{H}^{subposet}$ Hierarchy combine one of the ontologies in Figure 3 together with one of the ontologies in Figure 2 and one of the ontologies in Figure 1. In the following sections, we will explore how different ontologies in the $\mathbb{H}^{subposet}$ Hierarchy serve as design patterns.

4.2 Multimereology Hierarchy

Motivated by biomedical ontologies such as GALEN and Foundational Model of Anatomy, Bittner and Donnelly ([2],[3]) have investigated a class of ontologies that combine different kinds of mereological relations. In particular, they axiomatized three relations for part, component, and containment in an ontology which they call T_{fo_pcc}. The subtheory for the $part\_of$ relation has models which are isomorphic to a dense mereology with the weak supplementation principle. Models of the subtheory for the $component\_of$ relation are isomorphic to discrete mereologies which satisfy the weak supplementation principle as well as what Bittner and Donnelly refer to as the no-partial-overlap property – if x and y are distinct overlapping objects, then either x is a part of y or y is a part of x. Finally, models of the subtheory for the $contained\_in$ relation are isomorphic to a discrete partial ordering. Models of T_{fo_pcc} are amalgamations of the models of the three subtheories, and they are referred to as parthood-component-containment structures. Within the COLORE repository, these theories appear in the $\mathbb{H}^{multimereology}$ Hierarchy.

The ontology for parthood-component-containment structures also contains three axioms that specify how the substructures are combined. The component-of structure is a subordering of the parthood structure, while the relationship between containment and parthood satisfies the following two conditions – parts are contained in the container of the whole and that if a part contains something then so does the whole.

Bittner and Donnelly give an informal description of the models of their ontology, but do not provide a complete characterization of the models up to isomorphism. We can, however, use theories within the $\mathbb{H}^{subposet}$ Hierarchy to verify [6] that the models of the ontologies are isomorphic to the intended models of Bittner and Donnelly.

Theorem 2. *T_{fo_pcc} is definably equivalent to*
$(T_{tree\_mm\_mereology} \cup T_{dense\_weak\_separative} \cup T_{subposet})$
$\cup (T_{dense\_mm\_mereology} \cup T_{discrete\_mereology} \cup T_{lower\_preserve} \cup T_{upper\_preserve})$.

In this sense, we can prove that an ontology design pattern is correctly exemplified for given ontology O by proving that the core ontology is definably equivalent to O.

We can also use definable equivalence to extract multiple design patterns from the same ontologies in those cases where an ontology can be decomposed into modules.

[6] The proofs for all theorems can be found in
http://stl.mie.utoronto.ca/colore/subposet-theorems.pdf

Recognizing that T_{fo_pcc} is actually definably equivalent to two different ontologies in the $\mathbb{H}^{subposet}$ Hierarchy, we can specify two ontologies, T_{ppcmp} and T_{ppcnt}, which form a modular decomposition of T_{fo_pcc}.

Theorem 3. *T_{ppcmp} is definably equivalent to the ontology*
$T_{tree\_mm\_mereology} \cup T_{dense\_weak\_separative} \cup T_{subposet}$

Theorem 4. *T_{ppcnt} is definably equivalent to*
$T_{dense\_weak\_separative} \cup T_{discrete\_mereology} \cup T_{lower\_preserve} \cup T_{upper\_preserve}$

It is important to note that T_{ppcmp} and T_{ppcnt} are each definably equivalent to a unique ontology within the $\mathbb{H}^{subposet}$ Hierarchy. As stated in the previous section, each ontology within the $\mathbb{H}^{subposet}$ Hierarchy is a combination an ontology in the $\mathbb{H}^{ordering}$ Hierarchy, an ontology in $\mathbb{H}^{mereology}$ Hierarchy, and one of the "building block" ontologies in Figure 3 that specifies how the mereology and partial ordering are amalgamated.

4.3 Periods Hierarchy

The axioms in the ontologies of the $\mathbb{H}^{periods}$ Hierarchy[7] were first proposed by van Benthem in [1]. The key ontology of this hierarchy, referred to as T_{period}, constitutes the minimal set of conditions that must be met by any period structure and has two relations (*precedence* and *inclusion*) and two conservative definitions (for the *glb* and *overlaps* relations) as its signature. Transitivity and irreflexivity axioms for the precedence relation make it a strict partial order, and transitivity, reflexivity, and antisymmetry axioms for the inclusion relation make it a partial order; the axioms of monotonicity enforce correct interplay between the precedence and inclusion relations. Van Benthem further includes an axiom that guarantees the existence of greatest lower bounds between overlapping intervals.

Theorem 5. *T_{period} is definably equivalent to the ontology*
$T_{prod\_mereology} \cup T_{partial\_ordering} \cup (T_{upper\_preserve} \cup T_{lower\_reverse} \cup T_{chain\_antichain})$.

The relationships between the ontologies in this hierarchy were explored in [12]. In particular, additional theories within the $\mathbb{H}^{subposet}$ Hierarchy were shown to be definably equivalent to various extensions of T_{period} as axiomatized by van Benthem. This illustrates how we can use design patterns to specify the axiomatization of new ontologies in a hierarchy. Conversely, a subtheory of T_{period} was used to identify a new ontology within the $\mathbb{H}^{subposet}$ Hierarchy, thus illustrating how we can abstract new design patterns from a set of existing ontologies.

[7] http://code.google.com/p/colore/source/browse/trunk/ontologies/complex/periods

4.4 Subactivities in the PSL Ontology

The PSL Ontology uses the $subactivity$ relation to capture the basic intuitions for the composition of activities. This relation is a discrete partial ordering, in which primitive activities are the minimal elements.

The core ontology[8] $T_{subactivity}$ alone does not specify any relationship between the occurrence of an activity and occurrences of its subactivities. For example, we can compose $paint$ and $polish$ as subactivities of some other activity, say $surfacing$, and we can compose $make\_body$ and $make\_frame$ into another activity, say $fabricate$. However, this specification of subactivities alone does not allow us to say that $surfacing$ is a nondeterministic activity, or that $fabricate$ is a deterministic activity.

The primary motivation driving the axiomatization of T_{atomic} is to capture intuitions about the occurrence of concurrent activities. Since concurrent activities may have preconditions and effects that are not the conjunction of the preconditions and effects of their activities, concurrency in models of T_{atomic} is represented by the occurrence of one concurrent activity rather than multiple concurrent occurrences.

Atomic activities are either primitive or concurrent (in which case they have proper subactivities). The core ontology[9] T_{atomic} introduces the function $conc$ that maps any two atomic activities to the activity that is their concurrent composition. Essentially, what we call an atomic activity corresponds to some set of primitive activities – every concurrent activity is equivalent to the composition of a set of primitive activities. Although $T_{subactivity}$ can represent arbitrary composition of activities, the composition of atomic activities is restricted to concurrency.

Theorem 6. *The ontology* $T_{subactivity} \cup T_{atomic\_act}$ *is definably equivalent to the ontology*

$T_{cem\_mereology} \cup T_{discrete\_partial\_ordering} \cup T_{ideal}$

By this Theorem, models of $T_{subactivity} \cup T_{atomic\_act}$ are isomorphic to a structure in which a mereological field (on the set of atomic activities) forms an ideal within a discrete partial ordering (on the set of all activities).

4.5 Occurrence Trees in the PSL Ontology

Within the PSL Ontology, an occurrence tree[10] is a partially ordered set of activity occurrences, such that for a given set of activities, all discrete sequences of their occurrences are branches of the tree. An occurrence tree contains all occurrences of *all* activities; it is not simply the set of occurrences of a particular (possibly complex) activity. Because the tree is discrete, each activity occurrence in the tree has a unique successor occurrence of each activity.

In addition, there are constraints on which activities can possibly occur in some domain. Although occurrence trees characterize all sequences of activity occurrences, not all of these sequences will intuitively be physically possible within the domain.

[8] http://code.google.com/p/colore/source/browse/trunk/ontologies/complex/psl/subactivity

[9] http://code.google.com/p/colore/source/browse/trunk/ontologies/complex/psl/atomic

[10] http://code.google.com/p/colore/source/browse/trunk/ontologies/complex/psl/occtree

We will therefore want to consider the subtree of the occurrence tree that consists only of *possible* sequences of activity occurrences; this subtree is referred to as the legal occurrence tree.

Theorem 7. *The ontology* $T_{pslcore} \cup T_{occtree}$ *is definably equivalent to the ontology* $T_{tree\_mereology} \cup T_{tree} \cup T_{ideal}$

By this Theorem, models of $T_{pslcore} \cup T_{occtree}$ are isomorphic to a structure in which a tree mereology (on the set of legal activity occurrences) forms an ideal within a tree ordering (on the set of all activity occurrences). It is interesting to notice that T_{ideal} is used both for this ontology as well as for $T_{subactivity}$, demonstrating how one core ontology can be reused as a pattern across very different generic ontologies.

5 Discussion Points

In the previous section we provided examples of the ways in which core ontologies within COLORE can be utilised as CPs. We showed how a variety of real-world ontologies were comprised of core theories from the same hierarchy, and how even the same core theories were reused in different ontologies. We also demonstrated how core ontologies could be used to verify that an ontology contained the desired CPs (core theories), and how new core ontologies (CPs) could be identified by abstracting from ontologies in COLORE.

The ontologies in COLORE's hierarchies (specifically the core theories) correspond well to the definition of CPs provided by [6]: *"CPs are small ontologies that mediate between use cases (problem types) and design solutions. They are used as modelling components: ideally, an ontology results from a composition of CPs, with appropriate dependencies between them, plus the necessary design expansion based on specific needs"*. Based on this definition, each ontology in COLORE could be considered to be a CP as any of the modules could conceivably be reused to build other ontologies. However, in this paper we have focused on the core theories as they are most recognizable as CPs - although they are not necessarily domain-oriented, they are definably equivalent to theories that appear in multiple, different domains. They serve as syntactic templates for axioms in a variety of domains, so in a sense they combine aspects of CPs with the more domain-independent Logical OPs.

In this paper we have also explored the way in which the relationships defined in COLORE may be considered OPs, as the assistance they provide for ontology development is similar to the aid provided by OPs. Nevertheless, some of the features of COLORE offer capabilities beyond what is currently offered by the OP community. For example, because of the formalized nature of the relationships specified in COLORE, automated reasoning can be implemented to verify the mappings between the ontologies. In addition, the notion of reducibility can be implemented to identify useful CPs from ontologies in COLORE – the more theories that are reducible to a particular core theory, the more useful it is. Automated theorem provers may also be used to verify that an ontology is in fact a core theory. Lastly, OPs were not intended to be restricted to a particular representation language [6] and the use of first-order logic in COLORE supports this ideal as patterns from a wide range of languages may be represented.

We should emphasize that we do not believe that COLORE can or should replace traditional OPs. Although there are aspects of OPs for which COLORE offers similar solutions, there are also OPs which are completely absent from the relationships in COLORE. We believe that COLORE offers useful perspectives on OPs that may be beneficial to the OP community. In the other direction, much attention has been paid to promoting the use of OPs (CPs specifically); the results of this may be useful for the future development of COLORE. In particular, we can learn from the use of generalized use cases and competency questions to aid in users in the reuse of CPs [6, 5], as future plans for COLORE include the incorporation of competency questions as requirements [10] to identify suitable ontologies.

References

1. van Benthem, Johan F. A. K. (1991) *The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse*, Springer; 2nd edition.
2. Bittner, T. (2004) Axioms for parthood and containment relations in bio-ontologies, *Proceedings of KR-MED 2004: First International Workshop on Formal Biomedical Knowledge Representation*.
3. Bittner, T. and Donnelly, M. (2005) Computational ontologies of parthood, componenthood, and containment. Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence 2005 (IJCAI05). Kaelbling
4. Enderton, H.: Mathematical Introduction to Logic. Academic Press (1972)
5. Gangemi, A. (2005) Ontology Design Patterns for Semantic Web Content. *Proc. of International Semantic Web Conference*, pp. 262–267, Springer.
6. Gangemi, A., and Presutti, V. (2009) Ontology Design Patterns. In: *Handbook of Ontologies, 2nd Edition*, S. Staab (ed.), pp. 221–243, Springer.
7. Grüninger, M. (2009) Process Ontologies. In: *Handbook of Ontologies, 2nd Edition*, S. Staab (ed.), pp. 419–421, Springer.
8. Grüninger, M., Hahmann, T., Hashemi, A., and Ong, D. (2010) Ontology Verification with Repositories. *Proc. of the Sixth Int. Conference on Formal Ontologies in Information Systems (FOIS 2010)*, pp. 317–330, IOS Press.
9. Grüninger, M., Hahmann, T., Hashemi, A., Ong, D., and Ozgovde, A. (2012) Modular first-order ontologies via repositories, *Applied Ontology*, 7:169-210.
10. Katsumi, M. (2011) *A Methodology for the Development and Verification of Expressive Ontologies* (Master's Thesis). Retrieved from verb-https://tspace.library.utoronto.ca/bitstream/1807/31274/1/Katsumi_Megan_S_201111_MASc_thesis.pdf
11. Katsumi, M., and Grüninger, M. (2010) Theorem Proving in the Ontology Lifecycle. *Proc. of Knowledge Engineering and Ontology Design*
12. Ong, D. and Gruninger, M. (2011) Constructing an Ontology Repository: A Case Study with Theories of Time Intervals, Fourth International Workshop on Modular Ontologies. Ljubljana, Slovenia.
13. Scherp, A., Saathof, C., Franz, T. and Staab, S. (2011) Designing core ontologies, *Applied Ontology* 6:177-221.
14. Schorlemmer, M. and Kalfoglou, Y. (2008) Institutionalising Ontology-Based Semantic Integration, *Applied Ontology* 3:131-150.
15. Tsai, Hsing-chien (2009) Decidability of mereological theories, *Logic and Logical Philosophy* 18:45-63.

Ontology Design Patterns in Use
Lessons Learnt from an Ontology Engineering Case

Karl Hammar

Jönköping University
P.O. Box 1026
551 11 Jönköping, Sweden
karl.hammar@jth.hj.se

Abstract. Ontology Design Patterns show promise in enabling simpler, faster, more correct Ontology Engineering by laymen and experts alike. Evaluation of such patterns has typically been performed in experiments set up with artificial scenarios and measured by quantitative metrics and surveys. This paper presents an observational case study of content pattern usage in configuration of an event processing system. Results indicate that while structural characteristics of patterns are of some importance, greater emphasis needs to be put on pattern metadata and the development of pattern catalogue features.

Keywords: Ontology design pattern, evaluation, case study, event processing

1 Introduction

Ontology Design Patterns (ODPs) are intended to help guide domain experts in ontology engineering work, by packaging reusable best practice into small blocks of ontology functionality, either to be used as-is by practitioners, or to be used as inspiration and guidance for own development. This idea has gained some traction within the academic community, as evidenced by the WOP series of workshops held at ISWC 2009, 2010, and now 2012. If such patterns are to be accepted as useful artifacts also in practice, it is essential that they both model concepts and phenomena that are relevant to practitioners, and that they do so in a manner which makes them accessible and easy to use by said practitioners in real-world use cases.

This paper presents an attempt to learn more about ODP in-use qualities, how patterns are being used in a real case, and what users think about patterns, pattern portals, and pattern usage. In order to help formalize the study of the aforementioned issues, the following research questions have been selected for study:

1. What ODP characteristics do participants find helpful or harmful in ODP use?
2. How do users select and make use of ODPs?

3. What effects of ODP use on ontology engineering performance and resulting ontologies can be observed?

Within the ODP research community there exist different perspectives on what constitutes a pattern and how patterns should be categorized and sorted. The author of this paper largely subscribes to the definitions and pattern taxonomies presented within the NeOn project, published via the ODP portal[1]. The patterns mentioned and studied in this work are all examples of NeOn Content ODPs, consisting of both a pattern documentation and a reusable OWL building block.

The paper is structured as follows; Section 2 introduces some related work on ODP evaluation and Event Processing, Section 3 presents the project in which this case study took place, Section 4 describes the method employed, and Section 5 presents the findings and some recommendations based on these.

2 Related Work

The project that frames this case study concerns the development of a system for Semantic Complex Event Processing (SCEP) using ODPs as system configuration modules. The following sections present some background on CEP in general and on existing pattern evaluation work.

2.1 Complex Event Processing

Complex Event Processing (CEP) is introduced by Luckham & Frasca in [12]. In their approach, patterns based on temporal or causal links between events are defined and formalized into mapping rules. When executed over incoming time-indexed data streams, patterns connect lower level basic events to form higher level complex events. Luckham develops these ideas further in [11]. CEP has since been established as a useful method in many domains, and CEP based on sensor data feeds has been explored in many papers, using RFID sensors, cameras, accelerometers, etc.

As indicated by Anicic et al. in [2], most CEP approaches however have some drawbacks, particularly in terms of recognizing events using background knowledge. Only those relations between events and entities which are made explicit in the input data stream can be used for detection and correlation purposes. In order to overcome these limitations Anicic et al. suggest Semantic Complex Event Processeing (SCEP), in which background knowledge is encoded into knowledge bases that are accessed by a rules engine to support CEP.

2.2 ODP Evaluation

The effects of object oriented pattern use in software engineering and the harmful or beneficial properties of such patterns have been studied extensively, see for

[1] `http://ontologydesignpatterns.org`

instance [1, 13, 15]. Evaluations of pattern use in conceptual modeling is less common, but some examples of this type of research have been published, e.g. [14]. When it comes to pattern use in ontology engineering, as the author has previously found [8], the amount of work is also rather limited.

Possible benefits of ODP usage in ontology engineering have been shown by Blomqvist et al. in [3] and [4], both of which tested ODP usage according to the eXtreme Design method, by way of experimental setups with master and PhD student groups, and quantitative and qualitative surveys. Their results indicate that within this setting and for the modeled scenario, ODPs are perceived as useful, and the use of them result in fewer instances of a set of common modeling mistakes. However, they also report a perceived overhead associated with using the XD methodology and tooling, and find no strong support for ODPs improving the speed of ontology development. These experiments do not study the characteristics of the individual patterns in use.

Iannone et al. in [10] propose a semantics for expressing and method for computing the modularity (and consequently reusability characteristics) of ontology patterns. The method is implemented in a plugin for Protégé 4. The patterns under study are OPPL patterns and the algorithm presented is therefore incompatible with the view of ODPs as presented within the ODP community portal. However, conceptually the calculation of local and non local effects of pattern use seem to be relevant also for Content ODPs.

3 Case Characterization[2]

The project framing this case study is a small spinoff project from a larger project on threat detection using sensor systems where the partner research institute (hereafter RI) is involved. The work at RI focuses on development of a rule-based CEP subsystem intended to help isolate and correlate critical situations and threats based on incoming data. Within the spinoff project the aim is to develop the same functionality using semantic technologies. The motivation for this project is the increased flexibility of reasoning associated with using description logic languages, and the perceived gain in ease of reconfigurability associated with the use of ODPs. The following sections introduce the case participants and the architecture of the SCEP system.

3.1 Participants

Three participants attended the modeling workshops, participants A, B, and C. They are all male, and in the age bracket from 35 to 55 years. All three are researchers (two PhDs, one MSc) in software engineering or conceptual and data modeling within RI, and all three have some experience in such modeling. B and C have little or no prior knowledge of semantic web ontologies and

[2] For reasons of integrity and confidentiality, the case description and published data has been anonymized.

semantic technologies, whereas A has worked on these topics quite extensively, among other things researching rule languages for reasoning over semantic web ontologies. Their respective specialities are as follows:

- A has published on ontology matching, rule languages, model transformations, semantic technology use cases, etc.
- B has published on information logistics, mobile computing, context- and task-aware computing, etc.
- C has published on component based software engineering, middlewares, service orientation, system architectures, garbage collectors, etc.

3.2 System Architecture

The core of the system is a live observation knowledge base, defined according to an ontology schema. The ontology consists of both general features that are always relevant in the context of such a system (vocabularies of time, geographical locations and distances, sensor metadata, etc), and of features that are scenario and deployment specific. The latter features are imported from a set of four configuration knowledge bases, that together define system knowledge fusion behavior. These knowledge bases define, respectively: scenario configuration (i.e. background/context knowledge), situation correlation behavior, observation/entity fusion behavior, and critical situation detection behavior. Their contents are constructed by importing and adapting content ontology design patterns from a pattern repository.

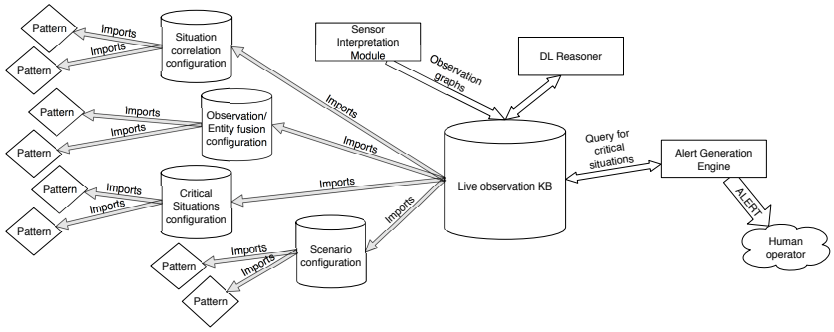


Fig. 1. Semantic Knowledge Fusion system architecture.

Data input from the deployed sensor subsystems is mapped to the general ontology vocabulary by sensor interpretation modules and stored as observation graphs in the knowledge base. A description logic reasoner is executed on the knowledge base and inferences about these observations are made. Then a rule engine is executed on top of the inferred knowledge, allowing greater expressivity in reasoning than that provided by description logic only (with the

rules employed being embedded within the utilized CODPs by way of annotation properties). If any observations are inferred to be instances of the critical situations defined within the critical situation configuration knowledge base, an alert is raised for a human operator to investigate the situation. For an overview of the system architecture, see Figure 1, and for a more in-depth description of the system see [9].

4 Method

Observation and data gathering was performed at a two-day modeling workshop at RI. The purpose of this workshop was to present the developments on the proposed system architecture and a prototype of the software to the participants, and to encourage them to develop configurations for it, thereby validating the applicability of the approach to their deployment scenarios.

Two scenario descriptions developed within the project were used to describe system deployment contexts[3]. The participants then attempted to model some typical relevant critical situations associated with each of these scenarios. Two examples of such critical situations are listed below:

- A gang is four or more people who have been seen together via at least three cameras over at least fifteen minutes and who are all wearing the same color clothing. A critical situation occurs when a gang of five or more football fans are loud and have within the last hour been spotted by a camera at a bar.
- Two vehicles are the same if they have the same license plate number *or* have the same brand, model and color and are observed by two cameras located at the same physical place within five seconds. A vehicle is behaving oddly if observed driving less than 15 km/h in three different cameras.

To their aid, the participants had a set of twenty ontology design patterns, of which fourteen were selected from the ODP community portal, and six were selected from other research projects. They were not provided with any training in pattern use, and were not recommended any particular development method, on the basis that providing such recommendations or training would restrict the participants' behavior and interaction with the patterns and the possibility of learning from their work.

During the modeling sessions data was gathered by way of audio and video recordings of the work in progress, photographs taken of ontology prototypes on the whiteboard, and notes taken on perceived key actions, behaviors, and trends taken independently by two researchers, the author and a senior professor with extensive experience of this research method. At the end of the second workshop day a semistructured group interview was held where the participants were queried about a number of different aspects of their experience and opinions on ODP use. Additionally, issues and statements of particular interest observed during the workshop were revisited and discussed, and conflicting interpretations resolved.

[3] Downloadable from `http://purl.karlhammar.com/data/wop2012/`

4.1 Data Analysis

Upon completing the workshop, the recorded material was transcribed into text. The vast majority of the material was immediately understandable. In the cases where ambiguities required interpretation, markers were put down. Those sections were revisited at the end of transcription, when a greater experience of the participants' voices was established, and in the majority of cases then resolved. The few uncertainties that remained were clearly marked out in the transcribed text, and subsequently ignored in later analysis steps.

Table 1. Codes used in data analysis.

Code	Label	Code	Label
1	ODP structure	3	ODP catalogue and selection
1.1	ODP size	4	ODP effects
1.2	ODP imports	4.1	Efficiency
1.3	ODP complexity	4.2	Usefulness
2	OE method observations	5	Pattern insufficient
2.1	ODP method observations	6	ODP usage prerequisites
2.1.1	ODPs-as-guidance	7	DL/semantics limitations
2.1.2	ODPs-as-error-control	8	Top-down/bottom-up choices
2.1.3	ODP-attributable errors	9	Existing implicit ontology effects
2.1.4	ODPs-as-ground ontologies	10	Method/metamodel adequacy
2.2	Modeling errors		

The text material (notes and transcripts) was then analyzed according to established transcript analysis methods [5, 7]. All the texts were read through and fragments coded by theme. The texts were read twice, once to establish coding categories in the material (see Table 1), and once to apply codes to the text corpus. The fragments were grouped by code, and the collected material pertaining to each code studied to see what conclusions could be drawn regarding participant experiences, opinions and behavior.

4.2 On Validity and Generalizability

Many methods of increasing validity and reliability in case studies and theories based on them have been proposed [6, 16]. A common approach in such methods is to limit the potential bias in data collection, coding, and analysis, by involving multiple researchers, to verify each other's analyses and data. Another recommendation is to involve case participants and to let them verify the perceived veracity of data and analyses. Yet another common approach is to triangulate results by using multiple data collection methods. In this study, two researchers were involved in data collection and note-taking. Multiple data collection methods were employed (audiovisual recordings of modeling sessions, researcher notes, interview transcripts). Preliminary analyses were verified against case partici-

pant opinion by way of a group interview. Due to resource limitations, coding and analysis was performed by only one person.

A downside to case studies is that the generalizability of results is limited. In fact, case study results not supported by similar results from other cases or by well-established theory cannot be said to be scientifically generalizable at all. That is not to say that these types of results are useless – on the contrary, if the characteristics of a studied case are similar to those of a new project in development, recommendations can often be reused from such results. However, there can be no *guarantees* of applicability made, no warranty of a causal link between specified behavior and some expected outcome, granted by the qualitative researcher. The author adheres to this perspective and makes no guarantees of replicability of the results, but is convinced that the ODP community will anyway benefit from the knowledge gained in this study.

5 Findings and Recommendations

The following sections describe the data gathered, and present some observations and analyses pertaining to the research questions garnered from said data. Some of the analyses are accompanied by brief recommendations for ODP researchers, based on what has been observed in this case.

5.1 Data

The resulting dataset comprises some 21600 words, or approximately 85 pages of text. Of these, 16 pages are researcher notes, and 69 pages are audio or video transcriptions. The participants were initially skeptical about being recorded on film, and their behavior changed noticeably when cameras were present, becoming quite a lot more formal and tense. In order to promote a good natural working environment for observations, the researchers chose to turn off the recording equipment initially, turning it back on only when the participants had gotten warmed up to the task and seemed less concerned about this. Due to the triangulation in analysis, this is believed to have little effect on the reliability of the results however.

Additionally, six whiteboard illustrations were photographed. There were 187 applications of codes to fragments, with the distribution of fragments over codes shown in Table 2.

5.2 Important ODP Features

During the modeling and subsequent interviews, the issues of ODP *size* and ODP *import count* were brought up.

The participants initially expressed divergent opinions regarding effect of OWL import statements in ODPs. Participant A considered imports quite helpful in that the reconciliation of imported base- or lower-level concepts with one's own model provided a good opportunity for validating the soundness of one's

Table 2. Distribution of fragments to codes.

Code label	Fragments	Code label	Fragments
ODP structure	1	ODP catalogue and selection	28
ODP size	3	ODP effects	3
ODP imports	10	Efficiency	11
ODP complexity	1	Usefulness	13
OE method observations	8	Pattern insufficient	12
ODP method observations	19	ODP usage prerequisites	11
ODPs-as-guidance	21	DL/semantics limitations	8
ODPs-as-error-control	7	Top-down/bottom-up choices	7
ODP-attributable errors	8	Existing implicit ontology effects	1
ODPs-as-ground ontologies	8	Method/metamodel adequacy	6
Modeling errors	1		

own design. He also emphasized the advantage of getting a foundational logic "for free" that one would not otherwise have had time to develop. Participant C expressed an understanding of the tension between reuse and applicability presented by the import feature and large import closures, comparing it to discussions in the OOP design pattern community in the nineties. Participant B criticized the use of imports on the grounds that the expansion of ODP size that such imports imply negatively affects ODP usability, and on the grounds that the base concepts included by imported patterns may be incompatible with one's own world view, being written for some other purpose:

> *"I really have to know what is there and what does it mean. And maybe it's written with some other focus, some other direction, some other goal. And I don't believe in this general modeling of the universe that fits all purposes."* – Participant B

Participant B also indicated that he would use the idea of a pattern as presented in a pattern catalogue and reimplement it, rather than reuse an existing OWL building block, if that block contained too many imports or dependencies. After some discussions Participant A agreed to the soundness of such a method in the case of a large import closure not directly relevant to the problem at hand. Both participants A and B proposed that a better solution would be to add support for partial imports to tools and standards.

In terms of the size of patterns, the participants emphasized during the interview session the importance of patterns being small enough to be easily understood in a minute or two of study. They considered an appropriate size to be three-four classes and the object- and datatype properties associated with them. They drew parallels to OO design patterns which are frequently of approximately this size. This expressed preference is consistent with the patterns they selected during modeling, all of which contained three or fewer classes.

Recommendations Avoid using imports in patterns unless the imported concepts or properties are necessary for pattern functionality. Support the development of partial import functionality in standards and tools. When possible, develop smaller rather than larger patterns.

5.3 Pattern Selection

It was observed by both researchers present that the single most important variable in ODP selection from the pattern catalogue was pattern naming. If a name "rang a bell" the participants proceeded with studying the pattern specifics to see whether the pattern was suitable in their case. This observation is supported by participant feedback at the interview session. The participants also suggested that description texts and competency questions (formalizations of design requirements as questions that the ODP is able to answer) were important selection criteria that should be emphasized in an ODP catalogue. Additionally, they considered the possible negative consequences of applying a certain pattern to a problem to be of particular importance in selecting and applying patterns.

On the subject of pattern catalogues, the participants indicated that they considered the two catalogues to which they had been exposed (the ODP community portal and the one developed for these sessions) to be unordered and unintuitive, holding patterns of varying completeness, abstraction level and domain, all mixed in one long list. The participants suggested that they would find it easier to navigate a catalogue that was structured according to topic, architecture tier, abstraction level, or some other hierarchy:

> *"You also know the old classification of upper ontologies, domain ontologies, and task ontologies. You know this old picture. This, at least this structure should be present."* – Participant A

Further participant suggestions for improvements to ODP catalogue usability included the addition of graphical illustration of pattern dependencies, and providing a semantic search engine across ODPs held in the catalogue. The former suggestion was inspired by an illustration from the Core J2EE Patterns web page[4] that the participants found helpful in deciphering pattern intent, and which Participant C in particular argued would be helpful in understanding the structure of a set of ODP patterns. The latter suggestion was that a search engine be added allowing users to search through concepts and properties present in ODPs in the catalogue, ideally including NLP techniques to match for synonyms and related terms.

Recommendations Ensure that pattern catalogues include complete and consistent pattern metadata, paying particular attention to pattern names, descriptions, competency questions, and negative effects. Ensure that pattern catalogues are structured according to a useful task- or abstraction-oriented hierarchy. Clarify interdependencies between patterns in pattern catalogues. Develop pattern catalogue search engines.

[4] `http://java.sun.com/blueprints/corej2eepatterns/Patterns/`

5.4 ODP Usage Method

The participants initially developed their designs on a whiteboard rather than on their computers. They used the patterns as guidance in development, rather than as concrete building blocks to be applied directly. When questioned on why this method of working was preferred, they stated that it was more flexible and required less commitment to a design in progress than immediately formalizing to OWL code. The participants would build a prototype solution to a whole problem in one go, rather than tackle one part of the problem at a time. This method is contrary to eXtreme Design, which emphasizes modular development and unit testing. However, the individual critical situations being modeled were rather small and self-contained, and it is uncertain whether this way of working would scale to larger and more complex problem spaces.

The guidance that the participants got out of the patterns appears to be of two types. To begin with, to the extent that patterns provided reasonable solutions to difficult to model problems, the pattern solutions were used as archetypes for own solutions on the whiteboard. This was the most common usage of patterns observed. In the second case, patterns were used to verify the correctness of modeling, by ensuring that the developed solution was consistent with the patterns selected:

> Participant B: *"Is a vehicle an agent?"*
> Participant A: *"Let's check the pattern!"*

The latter usage was observed both on the whiteboard and later on when attempting to formalize results into OWL files on a computer. In usage, the selected patterns were seen as optimal solutions to problems, and no reflections on the suitability of the patterns in question were observed. On the contrary, in some situations the participants attempted to realign their solutions to available design patterns even when this needlessly significantly increased the complexity of their solution. One example of this is the observed use of the *AgentRole* pattern in categorizing different types of staff, which in the scope of the problem could just as easily have been done via subsumption.

During modeling there were occasions when the work process slowed down, and the participants got caught up in discussions on how to define some very fundamental concepts such as situation, time, event, etc. When questioned, participants expressed a strong preference for such foundational concepts being available as patterns. While a few such foundational patterns have been extracted from DOLCE and made available in the community portal, their documentation is at the time of writing limited.

Recommendations Keep in mind the effects on pattern documentation requirements that arise when patterns are used as guides to development, rather than simply as building blocks. Ensure that common usage mistakes for individual patterns are clearly documented.

5.5 Effects of ODP Usage

Across the two days of working, a noticeable improvement in modeling speed among the participants could be observed. Tasks that in the morning took an hour to complete were in the afternoon performed in fifteen-twenty minutes. While this learning effect cannot be solely attributed to pattern use, the participants indicated that a certain efficiency gain is certainly due to them:

> *"I think it was helpful, it makes it clearer and furthers reuse, saving time."* – Participant A

This efficiency gain was most pronounced when the participants reused patterns which they had already tried once or twice on other problems. The participants also indicated that in order to get the most out of the design patterns, a practitioner needs to have developed some degree of familiarity with them:

> *"For me it's a new type of modeling [...] but it's understandable, and I can imagine if you know patterns, you are quite faster at inventing everything."* – Participant C

As has been mentioned in Section 5.4, the effects of ODP use on the process and resulting ontologies were not all beneficial. In some cases, over-dependence on patterns complicated the resulting ontologies needlessly, and misunderstanding of pattern documentation led to generally strange results. An example of the latter is the modeling of the characteristic "loudness", where the resulting model had time being loudness-indexed rather than the other way around. On the whole however these problems were minor compared to the observed and perceived benefits of ODP usage in guiding modeling.

6 Conclusions and Future Work

The users studied preferred small patterns over large ones, for reasons of understandability. They appreciated the foundational knowledge gained by large import closures, but found the consequent increase in pattern size troublesome, and would prefer partial import functionality if such were to be developed. Suggestions for pattern catalogue improvements include improving catalogue structure and search functionality, and increasing pattern documentation coverage. In order to decrease incorrect pattern usage, it is recommended that common pattern usage mistakes be documented. Finally, patterns were perceived as useful by the participants and the use of them was observed to increase the speed with which tasks were solved.

The author will in upcoming work attempt similar analyses in other cases, to study whether the results presented herein are found to apply to other projects in other domains also. Further, the author suggests that the ODP research community take under serious consideration the results presented herein that pertain to improvements of pattern catalogue structure, and would be happy to contribute to such work in the future.

References

1. Ampatzoglou, A., Chatzigeorgiou, A.: Evaluation of object-oriented design patterns in game development. Information and Software Technology 49(5), 445–454 (2007)
2. Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Stream Reasoning and Complex Event Processing in ETALIS. Semantic Web (2011)
3. Blomqvist, E., Gangemi, A., Presutti, V.: Experiments on Pattern-based Ontology Design. In: Proceedings of the Fifth International Conference on Knowledge Capture. pp. 41–48. ACM (2009)
4. Blomqvist, E., Presutti, V., Daga, E., Gangemi, A.: Experimenting with eXtreme Design. In: Knowledge Engineering and Management by the Masses. pp. 120–134. Springer (2010)
5. Burnard, P.: A method of analysing interview transcripts in qualitative research. Nurse Education Today 11(6), 461–466 (1991)
6. Eisenhardt, K.: Building Theories from Case Study Research. The Academy of Management Review pp. 532–550 (1989)
7. Garrison, D., Cleveland-Innes, M., Koole, M., Kappelman, J.: Revisiting methodological issues in transcript analysis: Negotiated coding and reliability. The Internet and Higher Education 9(1), 1–8 (2006)
8. Hammar, K.: The State of Ontology Pattern Research. In: Perspectives in Business Informatics Research: Associated Workshops and Doctoral Consortium (BIR 2011 Proceedings). Riga Technical University (2011)
9. Hammar, K.: Modular Semantic CEP for Threat Detection. In: Operations Research and Data Mining (ORADM) 2012 workshop proceedings. National Polytechnic Institute (2012)
10. Iannone, L., Palmisano, I., Rector, A., Stevens, R.: Assessing the Safety of Knowledge Patterns in OWL Ontologies. The Semantic Web: Research and Applications pp. 137–151 (2010)
11. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc. (2001)
12. Luckham, D., Frasca, B.: Complex Event Processing in Distributed Systems. Computer Systems Laboratory Technical Report CSL-TR-98-754. Stanford University, Stanford (1998)
13. Masuda, G., Sakamoto, N., Ushijima, K.: Evaluation and Analysis of Applying Design Patterns. In: Proceedings of the International Workshop on the Principles of Software Evolution (IWPSE99) (1999)
14. Mehmood, K., Cherfi, S., Comyn-Wattiau, I., Akoka, J.: A Pattern-Oriented Methodology for Conceptual Modeling Evaluation and Improvement. In: Research Challenges in Information Science (RCIS), 2011 Fifth International Conference on. pp. 1–11. IEEE (2011)
15. Prechelt, L., Unger-Lamprecht, B., Philippsen, M., Tichy, W.: Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance. IEEE Transactions on Software Engineering 28(6), 595–606 (2002)
16. Riege, A.: Validity and reliability tests in case study research: a literature review with "hands-on" applications for each research phase. Qualitative Market Research: An International Journal 6(2), 75–86 (2003)

Ontology Design Pattern Language Expressivity Requirements

Matthew Horridge[1], Mikel Egaña Aranguren[2], Jonathan Mortensen[1], Mark Musen[1], and Natalya F. Noy[1]

[1] Stanford Center for BioInformatics Research (BMIR), Stanford University, California, USA
[2] Biological Informatics, Centre for Plant Biotechnology and Genomics (CBGP), Technical University of Madrid, (UPM), Spain

Abstract. In recent years there has been a large amount of research into capturing, publishing and analysing Ontology Design Patterns (ODPs). However, there has not been any analysis into the typical language expressivity required to represent ODPs and how these requirements sit with lightweight fragments of the widely used ontology language OWL. In this paper we therefore present a survey on the language expressivity required to express the ODPs contained in the two main ODP catalogs: ODP.org and ODPS.sf.net. We surveyed a total of 104 machine processable ODPs and found that the OWL representations of these patterns typically require highly expressive fragments of the OWL language such as $\mathcal{ALCHIN}$, $\mathcal{SHOIN}$, $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$. We observed that most ODPs required the use of inverse properties, cardinality restrictions and universal restrictions, and that 10 patterns require OWL 2 constructs such as property chains, disjoint properties and qualified cardinality restrictions that are not available in OWL 1. Moreover, we found that most of the ODPs cannot be incorporated into ontologies that are constrained to fit into one of the OWL 2 profiles. Specifically, only 12 out of the 104 ODPs surveyed can be represented in OWL2EL, 13 in OWL2RL and 23 in OWL2QL. Despite this, we conjecture that it may be possible to rewrite and weaken some of them so that modellers using lightweight fragments of OWL can incorporate ODPs into their ontologies.

1 Introduction

Ontology Design Patterns (ODPs) are modelling solutions that solve recurrent ontology design problems [4]. Much of the work on ODPs has been inspired by the well known work on software design patterns from the mid nineties [3]; Gamma *et al.* presented and categorised small object-oriented models which are intended to be general solutions to specific but common problems in software design. The main benefit behind the use of software design patterns is that they decrease the incidence of poor modelling choices that could cause problems at a later date. Additionally, code that is based on software design patterns is more readable, more maintainable, and more reusable than code which is not. In many ways these are the exact reasons that motivate the use of design patterns as applied

to ontologies and in recent years there has been a thrust of research aiming at inventing and promoting the use of ODPs in ontology engineering.

Two prominent sources of ODPs are the `ontologydesignpatterns.org` catalog (ODP.org), and the `odps.sourceforge.net` catalog on the SourceForge.net website (ODPS.sf.net). The first is a centralised repository, initiated by Gangemi *et al.*, which takes submissions from researchers and practitioners that work in a variety of research areas and application domains. At the time of writing, this repository contains over 150 submissions binned into different categories. The second is based on the experience of Egaña *et al.* at the University of Manchester in representing biomedical knowledge. They propose 17 ODPs designed to tackle modelling problems and ontology language limitations. Both of these catalogs are specific to the Web Ontology Language OWL [10]. That is, ODPs are presented and described in terms of fragments of OWL ontologies.

The focus on OWL, as opposed to a "generic" (logic) based ontology language, is unsurprising. OWL is one of the most widely used ontology languages, it has excellent tool support in terms of editors and reasoners, and it is a World Wide Web Consortium (W3C) recommendation. The latest version of OWL is OWL 2, which became a W3C recommendation in October 2009 and is based on Description Logics. This logical underpinning provides a precise semantics and makes it possible to specify various reasoning tasks such as consistency checking, satisfiability testing and general entailment checking. Off the shelf automated reasoners such as ELK, FaCT++, HermiT, Pellet and Racer can be used to perform these key reasoning tasks and ontology development environments such as Protégé and the NeOn Toolkit provide hooks for integrating third party reasoners. Since OWL 2 is a highly expressive language key reasoning tasks like consistency checking have an extremely high worst case complexity: 2NExpTime-Complete [8], *i.e.*, intractable. This, coupled with the fact that implementing a highly optimised and scalable reasoner for the full language is a non-trivial task, and the fact that several well known biomedical ontologies, such as SNOMED [13], fit within smaller tractable fragments of the language, led to the development of the three so-called "profiles": OWL2EL, OWL2RL and OWL2-QL [9]. Each profile was carefully designed with practical use cases in mind, but a key aspect of each one is that it restricts what can be modelled so as to limit expressivity and make it possible, and easy, to implement efficient and scalable reasoners.

When it comes to applying ODPs to a domain or application ontology, and in particular an ontology which is specifically designed to fit into one of the OWL 2 profiles, it is not clear whether it is possible to incorporate any of the cataloged ODPs without compromising language expressivity constraints. This is a real concern, particularly for biomedical ontologies. For example, the large and well known medical ontology SNOMED is constrained to fit within the lightweight OWL2EL profile [12]. Many of the ontologies in the NCBO BioPortal repository [2] fall into the OWL2EL profile [6], and a sizeable number of them fall into the OWL2RL profile. Whether these ontologies were deliberately constrained to these profiles or not, it seems plausible that application of ODPs to these ontolo-

gies could easily take them outside of these profiles, thus losing the possibility of efficient, scalable reasoning. Ultimately, little is known about the language expressivity required to represent cataloged ODPs and how many ODPs can be represented or used within one of the OWL 2 profiles. The aim of this paper is therefore to present a survey and discussion of ODP language expressivity requirements.

2 Preliminaries

OWL 2 An OWL 2 ontology is a set of axioms (statements) which state something about the domain of interest. For example, a *subclass* axiom states that one class is a subclass of another class (*e.g.* Car is a subclass of Vehicle); an *inverse properties* axiom states that one property is the inverse of another property (*e.g.* hasPart is the inverse of isPartOf); and a *disjoint classes* axiom states that one class is disjoint with another class (*e.g.* Plant is disjoint with Animal). The OWL 2 language is underpinned by a highly expressive Description Logic called $\mathcal{SROIQ}$ [7]. This gives statements made in OWL a precisely defined meaning and, for a given ontology, makes it possible to use automated reasoning to compute whether or not a statement follows from the ontology. Statements that follow from an ontology are known as *entailments*. The process of reasoning used to determine whether or not an entailment follows from an ontology is known as *entailment checking*. Generally speaking, expressivity comes at a price—as expressivity increases so does the complexity (difficulty) of various reasoning problems such as entailment checking.

Description Logics As mentioned, OWL is underpinned by a Description Logic called $\mathcal{SROIQ}$. Generally speaking, Description Logics (DLs) are decidable fragments of First Order Logic. There are many different DLs, with each one being defined by the class, property and axiom constructors that it admits. One of the simplest DLs is known as $\mathcal{AL}$ (Attributive Language). This DL supports concept intersection (`owl:intersectionOf`), universal quantification (`owl:allValuesFrom`), limited existential quantification (`owl:someValuesFrom` with a filler restricted to `owl:Thing`) and atomic negation (`owl:complementOf` to named classes). More expressive DLs can be obtained from $\mathcal{AL}$ by adding further constructors. Each constructor is given a specific letter which is used to derive a name for any particular DL. For example, adding full negation $\mathcal{C}$ to $\mathcal{AL}$ produces the DL $\mathcal{ALC}$. Adding property hierarchy $\mathcal{H}$ (`rdfs:subPropertyOf`) to $\mathcal{ALC}$ produces $\mathcal{ALCH}$. Adding nominals $\mathcal{O}$ (`owl:oneOf`), inverse properties $\mathcal{I}$ (`owl:inverseOf`) and number restrictions $\mathcal{N}$ (`owl:minCardinality`, `owl:maxCardinality` or `owl:cardinality`) to $\mathcal{ALCH}$ produces $\mathcal{ALCHOIN}$. Finally adding transitive properties (`owl:TransitiveProperty`) to $\mathcal{ALCHOIN}$ produces $\mathcal{SHOIN}$, as the combination of $\mathcal{ALC}$ with transitive properties is abbreviated to $\mathcal{S}$. $\mathcal{SHOIN}$ is the DL that underpins OWL 1. OWL 2 extends the expressivity of OWL 1 with qualified cardinality $\mathcal{Q}$ to give $\mathcal{SHOIQ}$, and complex chains (`owl:propertyChainAxiom`), reflexive (`owl:ReflexiveProperty`), irreflexive (`owl:IrreflexiveProperty`), and disjoint

properties (owl:disjointWith) $\mathcal{R}$ to give $\mathcal{SROIQ}$. Distinct from the $\mathcal{AL}$ based family of DLs are $\mathcal{EL}$ based DLs. Rather than being based on universal quantification (owl:allValuesFrom), the $\mathcal{EL}$ family is based on existential quantification (owl:someValuesFrom), and universal quantification is prohibited. An important $\mathcal{EL}$ based DL is $\mathcal{EL}^{++}$ [1]. This is the DL which underpins OWL2EL and for which entailment checking can be efficiently performed in polynomial time w.r.t. the size of the input ontology.

OWL 2 Profiles As mentioned in the introduction, OWL 2 contains three profiles:

- OWL2EL—Based on the lightweight $\mathcal{EL}^{++}$ DL. OWL2EL is designed for representing large and moderately complex ontologies. In particular, it was designed with biomedical ontologies in mind.
- OWL2RL —This profile was designed to allow reasoning to be efficiently implemented with traditional rule engine based technologies.
- OWL2QL —Based on the lightweight DL-Lite family of DLs. This profile was designed for applications that combine a simple ontology with large amounts of instance data (possibly stored in a database).

Each profile limits the class, property and axiom constructors that it admits in order to achieve desirable properties in terms of reasoning. Of particular interest to this work is the OWL2EL profile, which is designed for representing large BioMedical ontologies. This profile prohibits the use of universal restrictions ($\forall$, owl:allValuesFrom), cardinality restrictions (owl:minCardinality, owl:maxCardinality, owl:cardinality), functional (owl:FunctionalProperty), inverse functional properties (owl:InverseFunctionalProperty), inverse properties (owl:inverseOf), disjunction ($\sqcup$, owl:unionOf), arbitrary negation ($\neg$, owl:complementOf), enumerations involving more that one individual (owl:oneOf), and disjoint, irreflexive and asymmetric properties. A full specification of each profile is beyond the scope of this paper—the interested reader is referred to the OWL 2 Profiles specification document [9].

Ontology Design Pattern Documents As mentioned in Section 1, the two main repositories of ODPs are the ODP.org repository and the ODPS.sf.net catalog. Each repository describes ODPs in a fairly standard way, *i.e.*, the name of the pattern, the problem the pattern is supposed to solve, limitations *etc.* Most of the patterns in each repository also contain a small ontology document that represents the pattern: a "pattern ontology". This pattern ontology document either (1) provides a domain specific example of the pattern, or (2) represents a small ontology that can be reused in the domain ontology. In the first case, the idea is that the pattern ontology document is copied into the domain ontology document with the appropriate translation of vocabulary from the example domain to the real domain. In the second case, the pattern ontology document is directly reused, by import, without modification. Patterns that fall into the second category are known as *Content* Design Patterns [11].

3 Materials and Method

Pattern Selection The ODPs that we considered for this study are the ones which contain a pattern ontology document as part of their pattern definition. That is, ODPs which are described with a parsable OWL ontology. Being able to parse a pattern (or an exemplar of a pattern) from an ontology document provides a clean way to automatically and unambiguously analyse the pattern, determine the expressivity of the language that is required to represent that pattern, and determine if this language falls into one of the three OWL 2 profiles. In the case of the ODP.org catalog, there are 91 proposed content ontology design patterns[3] that are represented as ontology documents. In the case of the ODPS.sf.net catalog[4] all 17 cataloged patterns are described with exemplar ontology documents.

Pattern Retrieval We accessed both catalogs on the 8th of August 2012. A total of 91 content patterns were listed in the ODP.org catalog. However, the ontology documents for the patterns PharmaInnova and BiologicalEntities could not be downloaded due to HTTP 404 (File Not Found) errors. Thus, we obtained a total of 89 pattern ontology documents from the ODP.org catalog. In the ODPS.sf.net catalog we retrieved all 17 pattern ontology documents. Out of these, we discarded the Normalisation and UpperLevelOntology pattern documents: these patterns do not require a fixed or minimum level of expressivity for instantiating them—the Normalisation pattern depends upon the class definitions that are appropriate to the domain being modelled, while the UpperLevelOntology pattern depends on the particular upper level ontology in question. This provided us with a total of 15 ontology pattern documents from the ODPS.sf.net catalog, and therefore a grand total of 104 pattern ontology documents for processing.

Pattern Processing and Analysis Each ontology document was parsed with the OWL API [5] (Version 3.3) in order to check that it was well formed. Next the OWL API Metrics and Profiles APIs were used to first compute the expressivity required to represent the pattern and then to check to see whether the pattern falls into OWL2DL and the three OWL profiles: OWL2EL, OWL2RL and OWL2QL. The results are summarised in the two tables in Section 4 below.

4 Results

Results are summarised in Table 1 and Table 2. The columns display the name of the pattern, the expressivity required to represent the pattern, whether the pattern fits into OWL2DL (DL), OWL2EL (EL), OWL2RL (RL) or OWL2QL (QL), whether the pattern contains universal restrictions ($\forall$), or disjunctions ($\sqcup$), and whether or not OWL 2 constructs are required to represent the pattern

[3] `http://ontologydesignpatterns.org/wiki/Category:ProposedContentOP`
[4] `http://odps.sourceforge.net/`

(Req.OWL2). Patterns that fulfil these latter seven properties are denoted with tick (✔) in the appropriate cell, otherwise the cell is left empty. Usage of universal restrictions (∀) and disjunction (⊔) has been singled out for presentation because these two constructors would otherwise be "lumped in" with the base language $\mathcal{AL}$. In addition to this, their use is prohibited in two out of the three OWL 2 profiles, namely OWL2EL and OWL2QL.

Table 1: Expressivity required for the ODPs from the ODP.org catalog.

Id	Name	Expressivity	DL	EL	RL	QL	∀	⊔	Req.OWL2
1	EthnicGroup	$\mathcal{EL}^{++}$	✔	✔		✔			
2	RTMSMapping	$\mathcal{EL}^{++}$	✔	✔		✔			
3	SpeciesConservation	$\mathcal{EL}^{++}$	✔	✔		✔			
4	Airline	$\mathcal{EL}^{++}$	✔	✔	✔	✔			
5	ConceptGroup	$\mathcal{EL}^{++}$	✔	✔	✔	✔			
6	ConceptTerms	$\mathcal{EL}^{++}$	✔	✔	✔	✔			
7	Metonymy	$\mathcal{EL}^{++}$	✔	✔	✔	✔			
8	SpeciesNames	$\mathcal{EL}^{++}$	✔	✔	✔	✔			
9	GoTop	$\mathcal{EL}^{++}$	✔	✔	✔				
10	Invoice	$\mathcal{ALF}(\mathcal{D})$	✔		✔				
11	Classification	$\mathcal{ALI}$	✔			✔			
12	Collection	$\mathcal{ALI}$	✔			✔			
13	CollectionEntity	$\mathcal{ALI}$	✔			✔			
14	Constituency	$\mathcal{ALI}$	✔			✔			
15	ActingFor	$\mathcal{ALI}$	✔		✔	✔			
16	PartOf	$\mathcal{ALI}+$	✔						
17	Place	$\mathcal{ALI}+$	✔						
18	Set	$\mathcal{ALI}(\mathcal{D})$	✔			✔			
19	Region	$\mathcal{ALI}(\mathcal{D})$	✔				✔		
20	Parameter	$\mathcal{ALI}(\mathcal{D})$	✔		✔	✔			
21	SpeciesEat	$\mathcal{ALEI}$	✔			✔			
22	AOS	$\mathcal{ALUHIF}+(\mathcal{D})$	✔					✔	
23	HasPest	$\mathcal{ALUHIF}+(\mathcal{D})$	✔					✔	
24	TimeInterval	$\mathcal{ALHN}(\mathcal{D})$	✔		✔				
25	ObjectRole	$\mathcal{ALHI}$	✔			✔			
26	Componency	$\mathcal{ALHI}+$	✔				✔		
27	Sequence	$\mathcal{ALHI}+$	✔						
28	IntensionExtension	$\mathcal{ALIN}$	✔				✔		
29	Situation	$\mathcal{ALIN}$	✔						
30	TimeIndexedSituation	$\mathcal{ALHIN}(\mathcal{D})$	✔						
31	LiteralReification	$\mathcal{ALHIN}+(\mathcal{D})$	✔				✔		
32	CommunicationEvent	$\mathcal{ALEHOIN}(\mathcal{D})$	✔				✔		
33	TypesOfEntities	$\mathcal{ALC}$	✔					✔	
34	AquaticResources	$\mathcal{ALCI}$	✔			✔			
35	Participation	$\mathcal{ALCI}$	✔			✔			
36	SpeciesHabitat	$\mathcal{ALCI}$	✔			✔			
37	InformationRealization	$\mathcal{ALCI}$	✔				✔		
38	GearVessel	$\mathcal{ALCI}$	✔		✔		✔		
39	RoleTask	$\mathcal{ALCI}$	✔		✔		✔		
40	TaskRole	$\mathcal{ALCI}$	✔		✔		✔		
41	AgentRole	$\mathcal{ALCHI}$	✔			✔			
42	GearSpecies	$\mathcal{ALCHI}$	✔			✔			
43	GearWaterArea	$\mathcal{ALCHI}$	✔			✔			
44	Communities	$\mathcal{ALCHI}$	✔				✔		
45	VesselSpecies	$\mathcal{ALCHI}$	✔				✔		
46	VesselWaterArea	$\mathcal{ALCHI}$	✔				✔		
47	Move	$\mathcal{ALCHI}$	✔		✔	✔			
48	CoParticipation	$\mathcal{ALCIN}$	✔						
49	CountingAs	$\mathcal{ALCIN}$	✔						
50	Price	$\mathcal{ALCIN}(\mathcal{D})$	✔						
51	SpeciesBathymetry	$\mathcal{ALCIN}(\mathcal{D})$	✔						
52	Criterion	$\mathcal{ALCHIN}$	✔						
53	CriterionSetter	$\mathcal{ALCHIN}$	✔						

Table 1: Expressivity required for the ODPs from the ODP.org catalog.

Id	Name	Expressivity	DL	EL	RL	QL	∀	⊔	Req.OWL2
54	Description	$\mathcal{ALCHIN}$	✔						
55	DescriptionAndSituation	$\mathcal{ALCHIN}$	✔						
56	ParticipationRole	$\mathcal{ALCHIN}$	✔						
57	Persons	$\mathcal{ALCHIN}$	✔					✔	
58	SpeciesConditions	$\mathcal{ALCHIN}$	✔						
59	TaskExecution	$\mathcal{ALCHIN}$	✔				✔		
60	Bag	$\mathcal{ALCHIN}(\mathcal{D})$	✔						
61	BasicPlanExecution	$\mathcal{ALCHIN}(\mathcal{D})$	✔				✔		
62	ClimaticZone	$\mathcal{ALCHIN}(\mathcal{D})$	✔						
63	NaryParticipation	$\mathcal{ALCHIN}(\mathcal{D})$	✔						
64	Observation	$\mathcal{ALCHIN}(\mathcal{D})$	✔						
65	ResourceAbundanceObs	$\mathcal{ALCHIN}(\mathcal{D})$	✔						
66	ResourceExploitationObs	$\mathcal{ALCHIN}(\mathcal{D})$	✔						
67	Tagging	$\mathcal{ALCHIN}(\mathcal{D})$	✔				✔		
68	TimeIndexedClassification	$\mathcal{ALCHIN}(\mathcal{D})$	✔						
69	TimeIndexedParticipation	$\mathcal{ALCHIN}(\mathcal{D})$	✔						
70	TimeIndexedPersonRole	$\mathcal{ALCHIN}(\mathcal{D})$	✔						
71	VerticalDistribution	$\mathcal{ALCHIN}(\mathcal{D})$	✔						
72	LinneanTaxonomy	$\mathcal{SHI}$	✔				✔	✔	
73	SimpleOrAggregated	$\mathcal{SHI}$	✔					✔	
74	ControlFlow	$\mathcal{SHIN}$	✔				✔		
75	InformationObjects	$\mathcal{SHIN}$	✔				✔	✔	
76	SimpleTopic	$\mathcal{SHIN}$	✔				✔		
77	Topic	$\mathcal{SHIN}$	✔				✔		
78	Action	$\mathcal{SHIN}(\mathcal{D})$	✔				✔		
79	BasicPlan	$\mathcal{SHIN}(\mathcal{D})$	✔				✔		
80	List	$\mathcal{SHIN}(\mathcal{D})$	✔						
81	PlanConditions	$\mathcal{SHIN}(\mathcal{D})$	✔				✔		
82	TimeIndexedPartOf	$\mathcal{SHIN}(\mathcal{D})$	✔						
83	PeriodicInterval	$\mathcal{SHOIN}(\mathcal{D})$	✔						
84	CatchRecord	$\mathcal{SHIQ}(\mathcal{D})$	✔				✔	✔	✔
85	Transition	$\mathcal{SHIQ}(\mathcal{D})$	✔				✔		✔
86	AquaticResourceObs	$\mathcal{SHOIQ}(\mathcal{D})$	✔				✔	✔	✔
87	Roles	$\mathcal{SRIN}$	✔						✔
88	SocialReality	$\mathcal{SRIN}$	✔						✔
89	Reaction	$\mathcal{SRIQ}$					✔		✔

ODP.org results summary Out of the 89 content ODPs 9 fit into the lightweight $\mathcal{EL}^{++}$ DL and therefore the OWL2EL profile. 13 fit into OWL2RL profile, and 22 into OWL2QL. One pattern, Reaction, violates the OWL2DL global restrictions which forces any ontology that includes this pattern out of OWL2DL. This particular pattern contains some disjoint properties axioms which specify that some properties which happen to be *Non-Simple* are disjoint with each other[5]. In terms of other prominent constructs, 77 patterns require inverse properties ($\mathcal{I}$), 45 patterns require cardinality restrictions of some form—either plain ($\mathcal{N}$) or qualified ($\mathcal{Q}$) min, max, or exact cardinality restrictions, 3 patterns require the implicit use of cardinality restrictions through the use of functional properties ($\mathcal{F}$), and 27 patterns use universal restrictions ($\forall$). Six patterns require OWL 2 constructs, that are not present in OWL 1, for their representation. Specifically, the Reaction pattern uses disjoint properties axioms and complex property chains, Roles uses complex property chains and anonymous inverse properties, and

[5] Non-Simple properties may not be used in certain positions in certain axioms, for example as operands in a DisjointProperties axiom. Roughly speaking, a property is Non-Simple if it is implied by a property chain (or transitive property).

SocialReality uses complex property chains. Four patterns (CatchRecord, Transition, Reaction and AquaticResourceObs) require the use of *qualified* cardinality restrictions ($\mathcal{Q}$), which are only available in OWL 2.

Table 2: Expressivity required for the ODPs from the ODPS.sf.net catalog.

Id	Name	Expressivity	DL	EL	RL	QL	$\forall$	$\sqcup$	Req.OWL2
1	NaryRelationship	$\mathcal{EL}^{++}$	✔	✔		✔			
2	CompositePropertyChain	$\mathcal{EL}^{++}$	✔	✔					✔
3	DefinedClassDescription	$\mathcal{EL}^{++}$	✔	✔					
4	NaryDataTypeRelationship	$\mathcal{ALEF}(\mathcal{D})$	✔						
5	Closure	$\mathcal{ALC}$	✔				✔		
6	EntityPropertyQuality	$\mathcal{ALCF}$	✔					✔	
7	ValuePartition	$\mathcal{ALCF}$	✔						
8	Selector	$\mathcal{ALCHF}$	✔					✔	
9	Exception	$\mathcal{ALCN}$	✔					✔	
10	EntityFeatureValue	$\mathcal{ALCQ}$	✔					✔	✔
11	InteractorRoleInteraction	$\mathcal{ALCQ}$	✔				✔	✔	✔
12	EntityQuality	$\mathcal{ALCIQ}$	✔				✔	✔	✔
13	AdaptedSEP	$\mathcal{S}$	✔					✔	
14	Sequence	$\mathcal{SHF}$	✔						
15	List	$\mathcal{SHN}$	✔					✔	

ODPS.sf.net results summary Out of the 15 pattern ontology documents, 3 fall into the OWL2EL profile, 0 fall into the OWL2RL profile, and 1 falls into the OWL2QL profile. One of these OWL2EL ontologies requires property chains, which are an OWL 2 construct. One of the patterns, EntityQuality, uses inverse properties ($\mathcal{I}$), 4 patterns use some form of explicit plain ($\mathcal{N}$) or qualified ($\mathcal{Q}$) cardinality restriction, while 5 ontologies require implicit cardinality restrictions due to the use of functional properties ($\mathcal{F}$). Only 3 patterns use universal restrictions ($\forall$), while 8 patterns use disjunction ($\sqcup$).

5 Analysis

Pattern Expressivity Requirements As can be seen from Tables 1 and 2, patterns from both the ODP.org and the ODPS.sf.net catalogs require a range of language expressivity, from the lightweight $\mathcal{EL}^{++}$ to the highly expressive languages $\mathcal{ALCHIN}$, $\mathcal{SHIN}$, $\mathcal{SHOIQ}$, and $\mathcal{SROIQ}$. Both catalogs lean towards requiring more expressive fragments of OWL, with many patterns that use constructs which bump up the expressivity from the base languages of $\mathcal{EL}^{++}$ or $\mathcal{AL}$. For example, in the ODP.org catalog it is typically the case that patterns require the use of inverse properties (77 out of 89 patterns) and cardinality restrictions (45 out of 89 patterns). It is also evident that both universal restrictions ($\forall$) and disjunctions ($\sqcup$) are sprinkled throughout the patterns in both patterns catalogs, with notable use universal restrictions in the ODP.org catalog, and disjunction within the ODPS.sf.net catalog. Universal restrictions are typically used to "close off" possibilities or model the local range of a property, whereas disjunctions are used to model "choices" or options for a property, so at first glance it makes sense that they appear in many patterns.

Pattern Expressivity and the OWL 2 Profiles While both pattern catalogs contain *some* patterns that can be represented within one or more of the OWL 2 profiles, it is clear that most of the patterns (59 out of 89 patterns from the ODP.org catalog and 12 out of 15 patterns from ODPS.sf.net catalog) cannot be represented in any languages corresponding to the profiles. This means that large swaths of patterns from both catalogs are "off limits" for ontology engineers targeting a specific profile.

Only 9 out of 89 and 3 out of 15 patterns from the ODP.org and ODPS.sf.net catalogs respectivelly can be represented in the OWL2EL profile language. One of the startlingly obvious reasons for this is that OWL2EL prohibits the use of inverse properties ($\mathcal{I}$), the use of cardinality restrictions ($\mathcal{N}$, $\mathcal{Q}$ or $\mathcal{F}$) and the use of universal restrictions ($\forall$). Interestingly, as far as the ODP.org catalog is concerned, more patterns fall into the OWL2RL and OWL2QL profiles than the OWL2EL profile. Unlike the OWL2EL profile, both of these profiles admit the use of inverse property axioms. Despite this OWL2QL is not strictly more expressive than OWL2EL, in fact it is a lightweight language profile that arguably puts more constraints on modellers than OWL2EL. This would seem to indicate inverse properties do play a major role in patterns violating the OWL2EL profile.

Pattern Expressivity and BioMedical Ontology Expressivity As mentioned previously, the OWL2EL profile is a pertinent profile for modelling and reasoning with biomedical ontologies. In fact, because of the prominence and importance biomedical ontologies OWL2EL was designed with these kinds of ontologies in mind. The language that underpins OWL2EL is expressive enough that it can be used to model typical biomedical ontologies, but its expressivity is limited to guarantee efficient reasoning. Indeed, OWL2EL reasoners such as ELK are able to classify large ontologies like SNOMED in a few seconds. To put things into perspective, more than half of the BioMedical ontologies contained in the NCBO BioPortal repository are OWL2EL ontologies [6], and the large medical ontology SNOMED expressly targets a fragment of this language in order to guarantee efficient reasoning [12]. It is therefore somewhat unfortunate that only a handful of ODPs can be expressed in OWL2EL. In essence, there is currently a tension between staying within a profile that offers fast and efficient reasoning and using ODPs. An interesting aspect of this tension is that patterns in the ODPS.sf.net catalog were specifically designed with biomedical ontology engineering in mind[6].

6 Towards Profile Friendly Patterns

Given the tension between typical biomedical ontology expressivity and the expressivity required to represent ODPs, an argument can be made in favour of producing versions of patterns that require limited expressivity. This argument also holds for other domains and the other profiles OWL2RL and OWL2QL. One

[6] It should be noted that the ODPS.sf.net catalog was compiled before OWL2EL was designed and published.

way of going about this would be to take existing patterns and rewrite or weaken them to conform to the required expressivity. In what follows we provide some examples of the ways in which the current ODP definitions could be modified to make them more usable with profile constrained ontologies.

Rewrite Cardinality Restrictions There is an abundance of patterns that required cardinality restrictions (45 out of 89 ontologies in the ODP.org catalog and 5 out of 15 in the ODPS.sf.net catalog). With both catalogs, it is plausible that cardinality restrictions were introduced into patterns either due to side-effects of particular tools, or due to modeller taste. This is a reasonable conclusion to arrive at because there are a high number of patterns which use min-one cardinality restrictions that could be directly replaced with existential restrictions without any loss of information[7]—in the case of the ODP.org catalog 24 patterns could be rewritten and in the case of ODPS.sf.net all 5 patterns which use cardinality restrictions could be rewritten.

Replace Universal Restrictions with Range Axioms Another construct that is prevalent throughout the patterns in ODP.org catalog but is not permitted in either OWL2EL or OWL2QL is the universal restriction ($\forall$). Upon casting an eye over patterns that use universal restrictions, it appears that many of them use these kinds of restrictions as simple local range constraints. For example, SubClassOf(A ObjectAllValuesFrom(hasPart B)) imposes a local range of B on the property hasPart for the class A. In patterns where the properties in these universal restrictions have limited usage (*i.e.* only one such universal restriction per property) it *may* be possible to replace the universal restriction with a global property range axiom (*i.e.*ObjectPropertyRange(hasPart B))[8]—these kinds of axioms are permitted in both OWL2EL and OWL2QL. Out of the 27 patterns from the ODP.org catalog that use universal restrictions, 23 patterns use them to impose one local range constraint on one property for one class. For these patterns it would be possible to write these local range constraints as global OWL range axioms without affecting the consistency and intended semantics of the pattern ontology.

Make Inverse Properties Optional The inverse properties axiom ($\mathcal{I}$) is prevalent throughout the ODP.org catalog and increases pattern expressivity leading to OWL2EL profile violation. From a philosophical point of view it is hard to say whether or not inverse properties are intrinsic to the ODPs in this catalog. However, an analysis of inverse property usage suggests that in some cases such usage could be due to the routine modelling practice of always declaring an in-

[7] The restriction ObjectMinCardinality(1 hasPart A) is semantically equivalent to the existential restriction ObjectSomeValuesFrom(hasPart A).

[8] Obviously, some care must be taken to ensure that multiple local ranges for a given class and property (possibly asserted in different patterns or in a domain ontology) do not produce an unsatisfiable range when converted to a global range and intersected with each other.

verse for a property[9]—irrespective of whether the inverse is used in a meaningful manner elsewhere in the ontology. As an illustration, consider that the axioms below appear in some pattern.

ObjectProperty(hasPart)
ObjectPropertyDomain(hasPart, A)
ObjectPropertyRange(hasPart, B)
...
ObjectInverseOf(hasPart, isPartOf)
ObjectPropertyDomain(isPartOf, B)
ObjectPropertyRange(isPartOf, A)

The property hasPart will be declared and used throughout the pattern, but 3 additional axioms will be also added: the inverse of hasPart (isPartOf) and the domain and range for this inverse as the "reverse" of the domain and range for hasPart (the primary property). In these cases, inverses could be made into an optional part of the design pattern that could be "bolted on" for domain or application ontologies that specifically require them, but left out for (biomedical) ontology engineers who want to remain with a profile such as OWL2EL. This optionality could be realised by offering different versions of the ODP or by splitting the ODP into separate ontologies which can be selected and imported by modellers only as required. Out of the 71 ontologies that contain inverse property axioms, 13 ontologies contain inverse usage as described above. A further 22 ontologies contain this "good practice" use of inverses plus asserting a property hierarchy for the inverses so that the inverse property hierarchy mirrors the primary property hierarchy. This hierarchy based kind of inverse usage could also be made optional.

7 Conclusions

In this paper we presented a study of the language expressivity required for using OWL ODPs. ODPs from the two main catalogs, ODP.org and ODPS.sf.net, were studied. Although there are a handful of patterns that can be represented using lightweight fragments of OWL most patterns require more heavyweight fragments containing inverse properties, cardinality restrictions, universal restrictions and disjunction. A small number of patterns require constructs that are only available in OWL 2, including property chains, disjoint properties and qualified cardinality restrictions. What is evident is that very few patterns can be represented in fragments of the language that are contained within one or more of the OWL 2 profiles. In particular, very few patterns, including ones specifically targeted at biomedical ontology construction, conform to the OWL2-EL profile. This means that there is a tension between using a language that was designed with biomedical ontologies in mind and using design patterns that were

[9] Often perceived as a good practice.

designed for biomedical ontologies. More generally, since all three OWL profiles were designed with the goal of supporting fast and efficient reasoning, modellers must currently make a choice between taking advantage of ODPs or taking advantage of high performance tools. An initial analysis of the constructs which lead to the high expressivity requirements of patterns suggest that some of these issues could be dealt with by rewriting patterns to use different constructs and making parts of patterns, especially those that use inverse properties, optional. As future work, it would be interesting to assess the impact of such changes on ontology engineering.

Acknowledgements This work was supported by the NIH Grants GM086587, HG004028, and LM007033. Mikel Egaña Aranguren is funded by the Marie Curie-COFUND Programme (FP7) of the European Union.

References

1. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *IJCAI 05, Edinburgh, Scotland, UK, July 30-August 5, 2005*, 2005.
2. Natalya F. Noy et al. BioPortal: Ontologies and integrated data resources at the click of a mouse. *Nucleic Acids Research*, 37(suppl 2):W170–W173, May 2009.
3. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
4. Aldo Gangemi and Valentina Presutti. *Ontology Design Patterns*. International Handbooks on Information Systems. Springer, 2009.
5. Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, February 2011.
6. Matthew Horridge, Bijan Parsia, and Ulrike Sattler. The state of biomedical ontologies. In *BioOntologies 2011 15th–16th July, Vienna Austria*, 2011.
7. Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *KR 2006, Lake District, United Kingdom*, pages 57–67. AAAI Press, June 2006.
8. Yevgeny Kazakov. $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In *KR 2008, Sydney, Australia, September 16-19, 2008*.
9. Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles. W3C Recommendation, W3C – World Wide Web Consortium, October 2009.
10. Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language structural specification and functional style syntax. W3C Recommendation, W3C – World Wide Web Consortium, October 2009.
11. Valentina Presutti and Aldo Gangemi. Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies. In *Proceedings of the 27th International Conference on Conceptual Modeling*, ER '08, pages 128–141, Berlin, Heidelberg, 2008. Springer-Verlag.
12. Kent A. Spackman. An examination of OWL and the requirements of a large health care terminology. In *OWLED 2007*.
13. Kent A. Spackman and Keith E. Campbell. SNOMED RT: A reference terminology for health care. In *In Proc. of AMIA Annual Fall Symposium*, 1997.

Modest Use of Ontology Design Patterns in a Repository of Biomedical Ontologies

Jonathan M. Mortensen, Matthew Horridge, Mark A. Musen, and Natalya F. Noy

Stanford Center for Biomedical Informatics Research
Stanford University, Stanford CA 94305, USA

Abstract. Ontology Design Patterns (ODPs) provide a means to capture best practice, to prevent modeling errors, and to encode formally common modeling situations for use during ontology development. Despite the popularity of ODPs and supposed positive effects from their use, there is scant empirical evidence of their level of adoption in real world ontologies or on their effectiveness. Knowing the goals of ODPs, they may assist in the development of large-scale biomedical ontologies. Before studying ODP effectiveness and applicability, we ask the following questions to understand better the landscape of ODP use: Are ODPs used in biomedical ontologies? Which patterns do the ontology developers use? In which ontologies? How frequently are patterns used? To answer these questions, we determined the adoption of ODPs from two popular ODP libraries among the ontologies in BioPortal, a large ontology repository that contains over 300 biomedical ontologies. We encoded 68 ODPs from two online libraries in the Ontology Pre-Processor Language, and, using these encodings, determined ODP prevalence in BioPortal ontologies. We found modest use of ODPs, with 33% of the ontologies containing at least one pattern. `Upper Level Ontology`, `Closure`, and `Value Partition` were the three most commonly used patterns, occurring in 20%, 9%, and 6% of the BioPortal ontologies, respectively. The low prevalence of ODPs may be due to lack of proper tooling, lack of user knowledge of and education about them, the age of the ontologies in the repository, or the specificity of some ODPs. We noted that there is a tension between the high expressivity of many ODPs and the goal of maintaining low expressivity of some biomedical ontologies. Additional tooling is necessary to make ODPs more accessible to domain experts. Furthermore, we suggest that ODPs may be developed in a bottom-up fashion, much like software-design patterns. [1]

Keywords: OWL, biomedical ontologies, BioPortal, Ontology Design Pattern, Ontology Pre-Processor Language

1 Ontology Design Patterns

There is a large body of research establishing and creating Ontology Design Patterns (ODPs) [11, 5]. Yet, there is little work to determine their use or effectiveness. In biomedicine, the development and use of ontologies are growing rapidly. This

[1] Accompanying online resources at http://www.stanford.edu/people/mortensen/odp

development process can be difficult and/or error prone. As such, ODPs would likely assist with this development process. In this study, as initial work in evaluating the effectiveness and applicability of ODPs in biomedical ontologies, we examine the prevalence of ODPs in a large corpus of ontologies related to biomedicine.

1.1 ODPs and ODP libraries

Software Design Patterns emerged in the 1990s, capturing recurring software design techniques seen in software [10]. Following a similar motivation, the Semantic Web community developed ODPs to alleviate some of the complexities in developing ontologies. ODPs, defined as "a modeling solution to solve a recurrent ontology design problems" [11], capture best practice and common modeling situations. The developers of ODPs suggest that by using the patterns, one can more easily avoid modeling errors, improve ontology quality, maintainability, and reuse [3].

ODPs have become quite popular recently, with multiple workshops held at ISWC, including one during ISWC 2012. There are two online catalogs of ODPs, the Manchester ODPs Public Catalog for bio-ontologies (MBOP) and OntologyDesignPatterns.org (ODP-Wiki) [9, 1]. These catalogs describe each pattern by the problem that it solves, the proposed solution, and the formal representation by which to instantiate the pattern. MBOP contains 17 patterns derived from its authors' experience in modeling ontologies in the biomedical domain and working with OWL-based ontologies in general. ODP-Wiki is a crowd-sourced effort to create an ODP library. The website owners ask for pattern submissions and then a committee reviews these submissions for approval. The approved patterns are then noted as such online. As of this writing, the committee has not approved any patterns but there are over 150 submissions.

Most of the submissions on ODP-Wiki are "content" ODPs. However, the site categorizes many other different types of ODPs. ODP-Wiki includes "structural" (methods to workaround for language expressivity limitations or define ontology shape/structure), "content" (modeling solutions for a specific domain), "correspondence" (methods to re-engineer an ontology to a different form or map an ontology to another), "reasoning" (patterns that enable one to obtain desired reasoning results),"presentation" (good practices for readability and usability), and "lexico-syntactic" (mapping linguistic structures to ontology entities) patterns—a categorization based on descriptions by Gamgemi and colleagues [11]. MBOP categorizes patterns as "extension" (workarounds for language expressivity limitations), "good practice" (good modeling practice) and "domain modeling" (solutions specific to certain domains). The "structural" classification encompasses the majority of the MBOP patterns. In this work, the structural and content ODPs are most relevant. Structural patterns are either logical, adding logical expressions not contained directly in the ontology language, or architectural, defining the structure/hierarchy of the ontology itself. Content ODPs model a specific domain situation, and are directly re-usable (i.e., they should be directly imported into an ontology and used). We omit lexico-syntactic, presentation, reasoning, and correspondence patterns from this work, as we cannot test for them using our framework.

Accompanying the MBOP, the Manchester group also developed the Ontology Pre-Processing Language (OPPL), both a language based on the Manchester syntax for OWL, and a software library, which leverages the OWL-API [14]. OPPL provides a way to manipulate ontologies, query for ODPs and instantiate them [16, 15, 2].

1.2 Biomedical Ontologies

In biomedicine, ontology use is rapidly increasing [7, 21]. For example, the National Center for Biomedical Ontology's BioPortal,[2] a repository of biomedical ontologies, contains over 300 ontologies and controlled terminologies as of this writing [18]. Biologists use biomedical ontologies to manage the large amount of data. Hospitals and related entities use them in the process of recording information about clinical encounters, during clinical decision support, billing, and so on. Because biomedical ontologies are often large and complex, developing them and ensuring that they conform to best practices poses a formidable challenge. Even the widely used ontologies frequently contain modeling errors. For instance, Rector and colleagues discovered modeling issues in SNOMED CT, one of the most widely used biomedical ontologies [19]. Researchers have found modeling errors in the National Cancer Institute thesaurus [8]. ODPs may be especially important in assisting with the challenge of modeling the large and complex biomedical domains while preventing errors. Before assessing the effect of using ODPs on the biomedical ontology modeling process, we first find the prevalence of ODPs in a large biomedical ontology corpus.

2 Methods

We quantified the use of ODPs from both MBOP and ODP-Wiki in BioPortal using OPPL and the OWL API. We first encoded ODPs in OPPL and validated their correctness (1) by using an expert opinion and (2) by comparing them to the examples in the library that served as a gold standard. We then obtained the ontologies from BioPortal, removing cases by use of predefined filtering criteria (See section 2.2). We normalized the ontologies to remove any differences in how they were specified, and then checked both the normalized and the original version for each encoded pattern, first filtering out patterns that cannot be represented in the ontology because it lacks the proper relations.

2.1 Pattern Selection

We used the following criteria to select the set of patterns for this study: The pattern must be (1) detectable, (2) non-trivial (that is, not just a template), (3) positively reviewed (if a review is available), and (4) available in a public catalog (in our case, either MBOP or ODP-Wiki). We use these criteria for the following reasons:

1. Using only detectable patterns may seem obvious; however, there are many patterns such as `n-ary relations`, or re-engineering patterns that cannot be detected without more information than just the ontology.
2. A template style pattern may not *require* the presence of any particular elements. Thus, it would be trivially present even if the ontology contained no elements of the pattern.
3. When available, we considered review information on ODP-Wiki. Poorly reviewed patterns may not yet be refined, making them difficult to encode, especially if they have a logical error.
4. We chose only publicly available patterns, as it is a necessary condition for both reproducibility of this study and the expectation of pattern re-use.

[2] http://bioportal.bioontology.org

Applying the criteria above to MBOP and ODP-Wiki, produced the following results:

- From the 17 patterns in MBOP, we used 15. The remaining 2 were undetectable
- From the 150 patterns in ODP-Wiki, we used 53. The remaining patterns were either optional or not positively reviewed.

Thus, we selected 68 patterns of 167.

2.2 Ontology Selection

From the available ontologies in BioPortal, we selected those ontologies that were publicly available, parseable, locatable (a file was easily obtainable), non-retired, available as a single file, and available as either OWL or OBO format. Applying these criteria to the 312 ontologies that were available in BioPortal as of January 2012, resulted in a set of 256 ontologies.

2.3 Pattern Encoding

OPPL and the OWL API are open-source standard libraries available to work with ontology design patterns and ontologies. We encoded the MBOP and ODP-Wiki patterns with OPPL. Some patterns could not be encoded in OPPL. Those patterns we encoded directly in Java using the OWL API. An example OPPL encoding of the `Value Partition` pattern (a way to specify a set of disjoint qualities the describe a concept) follows:

```
?v1:CLASS, ?v2:CLASS, ?param:CLASS
SELECT
ASSERTED ?param EquivalentTo ?v1 or ?v2,
ASSERTED ?v1 DisjointWith ?v2
BEGIN
ADD ?v1 subClassOf Thing
END;
```

In order to reduce computational complexity, we pruned pattern–ontology pairs by first checking whether the ontology contains the specific relationships between concepts that a given ODP requires. An ontology without those relationships cannot have the pattern as the catalog specifies it. Furthermore, for those patterns that could not occur in any ontology from our selection, based on the required relationships, we did not encode the pattern. In particular, many content patterns refer to specific relationships in the ontology. For example, according to ODP-Wiki, the pattern `Part Of` requires the relationship “isPartOf”. Thus, if an ontology does not have this relationship “isPartOf”, we know that it will not have the pattern. When searching, we disregard the namespace of any given pattern, in case the pattern simply uses a different namespace (i.e., we only match on the URI fragment, not including the namespace). One might consider searching with possible lexical variants of this relationship term to ensure one finds occurrences which capture the intension of the specified relationship. However, the point at which a given string no longer matches the initial string is not well defined. Furthermore, content ODPs directly import a small module, thus the relation should not vary across ontologies.

Table 1. Transforms applied exhaustively to an ontology to normalize it.

Axiom	Transformation
prop min 1 C	prop some C
prop exactly n C	prop min n C, prop max n C
prop value i	prop some i
Property in Anonymous Class	Simplify Property (Removing inverses) and re-insert
C1 and (C2 and C3)	C1 and C2 and C3
C1 or (C2 or C3)	C1 or C2 or C3
C1 EquivalentTo C2	C1 SubClassOf C2, C2 SubClassOf C1
C1 DisjointUnionOf C2 ... Cn	DisjointClasses: C2 ... Cn, C1 EquivalentTo (C2 ... Cn)
C1 or ... or Cn SubClassOf D1 and ... and Dn	C1 SubClassOf D1 ... Cn SubClassOf D1 ... C1 SubClassOf Dn ... Cn SubClassOf Dn
DisjointClasses: C1 ... Cn	Ci DisjointWith Cj for 1 <= i <j <=n

Finally, during encoding, to gather additional information about a pattern, we noted the OWL 2 description logic constructs each utilizes. We note these because certain constructs have higher expressivity requirements. Higher expressivity comes at a higher computational cost. This fact may provide insight as to why biomedical ontologies instantiate only certain patterns.

2.4 Normalization

When specifying an ontology, one can represent the same conceptualization using different language constructs. This phenomenon is particularly common in OWL. To prevent missing a pattern that may be specified in a slightly different language than that found in an ontology, we applied pattern detection to a normalized version of each ontology as well. Table 1 lists the normalizations that we performed. These transformations follow from the OWL 2 specification and convert 'syntactic sugar' conventions provided by the language to a standard form. For example, applying the transformations to `Dog EquivalentTo Canine` results in `Dog SubClassOf Canine` and `Canine SubClassOf Dog`. While one could use a reasoner to infer equality of certain constructs, this was not computationally feasible in this study.

2.5 Computation

After creating a list of encoded patterns, and of original and normalized versions of all ontologies in the study, we searched for each pattern in the ontologies using the encodings and the software libraries. Checking for 68 patterns in nearly 300 ontologies, some of which have tens of thousands of classes, is a computationally intensive task. A brute-force approach to this task was unreasonable, as a single run through all possible ontology–pattern pairs would have taken over a week. We performed a few optimizations to speed up the runtime of the experiment. First, as described before in section 2.3, we pruned the patterns first by searching whether or not the ontology contains the necessary relations. Next, we created a specialized cache to store computationally intensive search operations that are shared across multiple pattern

queries per ontology. Finally, we distributed the pattern search process to a large cluster of 20 nodes, each with 24 cores and 96GB of RAM. We do not attempt to count the number of times a pattern occurs in a given ontology, as doing so also increases complexity. We limit the running time of any ontology–pattern pair to one week. During our analysis, approximately 150 pairs did not complete. These were complex patterns and complex ontologies (e.g., FMA and `Normalization`).

2.6 Validation

A pattern that is encoded (incorrectly) in a way that is too general could lead to a false positive, matching any ontology. For example, encoding `Value Partition` without the EquivalentClass component would match any ontology that has disjoint classes, which is not a true instantiation of `Value Partition`. An incorrectly specified pattern would lead to a false negative, not matching any ontology even though the pattern is present. We therefore verified that the patterns were correctly encoded both by involving an expert and by comparing them against a reference standard. The pattern encodings were manually inspected by an author of this paper (MH). Second, where possible, we tested the patterns and software against reference examples provided by the catalogs. We expected to find each encoded pattern in its reference example. MBOP provided example OWL ontology implementations of every pattern. Because ODP-Wiki required the import of a specific ontology, the verification was trivial.

3 Results

Of the 68 patterns in the study, we encoded 8 patterns in OPPL, 5 with the OWL API, manually verified 7, and pruned 47 content ODPs that contained relationships that were not present in any of the ontologies in the study. These relationships included, for example, “isRegionFor”, “hasRTMSCode”, “participatingInEvent”. One of the patterns MBOP specifies is the `Upper Level Ontology` pattern. MBOP describes the pattern as good practice that allows one to integrate different ontologies in a grounded framework. To find the `Upper Level Ontology` pattern, we checked whether each ontology imported a unique upper-level concept from either of the following upper ontologies: DOLCE [12], BFO [13] or SUMO [17]. We manually verified the remaining 7 patterns that matched the necessary relations for a pattern. These patterns need not necessarily import the pattern, but only capture its intension (e.g., GALEN). For brevity, Table 2 presents only a list of the positive patterns and what ontologies instantiated them. We found that 14 patterns were present in at least one ontology. 33% of the OWL and OBO format ontologies in BioPortal contain a pattern, with `Upper Level Ontology` (20%), `Closure` (9%), and `Value Partition` (6%) most common. Other commonly used patterns include `Normalization` and `Composite Property Chaining`. Thirty ontologies included more than one pattern. The Ontology of Biomedical Investigations included the most patterns: `DefinedClassDescription`, `Closure`, `Value Partition`, `Sequence` and `Upper Level Ontology`.

Table 2: Patterns and the ontologies that instantiate them.

Composite Property Chaining (7 Ontologies)	
Model a double chain of properties, i.e. two chains that link four individuals.	
Brucellosis Ontology	SemanticScience Integrated Ontology
Infectious Disease Ontology	Skin Physiology Ontology
Influenza Ontology	SNOMED CT
RNA ontology	
Adapted SEP (1)	
Properly propagate partomony	
BioTop	
Closure (23)	
Simulate the Closed World Assumption in a concrete class	
Amino Acid	Kinetic Simulation Algorithm Ontology
BioAssay Ontology	Lipid Ontology
BioTop	NanoParticle Ontology
Bleeding History Phenotype	Neomark Oral Cancer-Centred Ontology
Bone Dysplasia Ontology	Ontology for Biomedical Investigations
Breast Cancer Grading Ontology	Ontology for disease genetic investigation
Cancer Research and Management ACGT Master Ontology	Ontology for Genetic Interval
Cognitive Atlas	Skin Physiology Ontology
DIKB-Evidence-Ontology	Subcellular Anatomy Ontology (SAO)
Gene Regulation Ontology	Suggested Ontology for Pharmacogenomics
IMGT-ONTOLOGY	Vaccine Ontology
Infectious Disease Ontology	
Defined Class Description (6)	
Create If-Then structures in OWL DL	
Adverse Event Reporting ontology	Ontology for Biomedical Investigations
Cancer Research and Management ACGT Master Ontology	Ontology for Drug Discovery Investigations
NanoParticle Ontology	SysMO-JERM
Value Partition (16)	
Model values of non-overlapping attributes exhaustively	
Adverse Event Reporting ontology	Influenza Ontology
Basic Formal Ontology	OBOE
Basic Vertebrate Anatomy	Ontology for Biomedical Investigations
BioTop	Ontology for disease genetic investigation
CAO	Ontology for Genetic Interval
Computer-based Patient Record Ontology	RNA ontology
General Formal Ontology	Situation-Based Access Control
Infectious Disease Ontology	Vaccine Ontology
Normalization (14)	

Ensure maintainability and explicit semantics by allowing polyhierarchy only through inference

Basic Formal Ontology	PMA 2010
Human developmental anatomy, abstract version	Proteomics Pipeline Infrastructure for CPTAC
IMGT-ONTOLOGY	SemanticScience Integrated Ontology
Mouse gross anatomy and development	Traditional Medicine Constitution Value Set
NIFSTD	Traditional Medicine Meridian Value Sets
OBO relationship types	Traditional Medicine Other Factors Value Set
Pilot Ontology	Traditional Medicine Signs and Symptoms Value Set

Upper Level Ontology (53)

Create ontologies with a consistent, philosophically grounded upper ontology

Adverse Event Reporting ontology	Mental Functioning Ontology
Basic Formal Ontology	Mosquito insecticide resistance
Basic Vertebrate Anatomy	NanoParticle Ontology
BioModels Ontology	Neomark Oral Cancer Ontology
BIRNLex	Neural ElectroMagnetic Ontologies
Bone Dysplasia Ontology	NIF Cell
Brucellosis Ontology	NIF Dysfunction
Cancer Chemoprevention Ontology	NIFSTD
Cancer Research and Management ACGT Master Ontology	NMR-instrument specific component of metabolomics investigations
CAO	Ontology for Biomedical Investigations
Cardiac Electrophysiology Ontology	Ontology for disease genetic investigation
Chemical Information Ontology	Ontology for Drug Discovery Investigations
Cognitive Paradigm Ontology	Ontology for General Medical Science
Computer-based Patient Record Ontology	Ontology for Genetic Interval
Drosophila development	Ontology for Parasite LifeCycle
eagle-i research resource ontology	Ontology of Data Mining
Electrocardiography Ontology	Ontology of Glucose Metabolism Disorder
FGED View	Ontology of Medically Related Social Entities
Gene Regulation Ontology	Phenotypic quality
General Formal Ontology	RadLex
General Formal Ontology: Biology	RNA ontology
Host Pathogen Interactions Ontology	Skin Physiology Ontology
IEDB View	Sleep Domain Ontology
Infectious Disease Ontology	Subcellular Anatomy Ontology (SAO)
Influenza Ontology	Translational Medicine Ontology
Information Artifact Ontology	Vaccine Ontology
Interaction Network Ontology	

Agent Role (1)

Represent agents and the roles they play

ICPS Network

Classification (1)

Represent the relations between concepts and entities to which concepts can be assigned

ICPS Network	
Componency (1)	
Represent (non-transitively) that objects either are proper parts of other objects, or have proper parts	
Computational Neuroscience Ontology	
Object Role (2)	
Represents objects and the roles they play	
ICPS Network	International Classification for Nursing Practice
Part Of (2)	
Represent entities and their parts.	
Ontology for Genetic Interval	SysMO-JERM
Place (2)	
Talk about places of things	
Galen	International Classification for Nursing Practice
Sequence (4)	
Model a sequence of events occuring one after another	
FGED View	Ontology for Biomedical Investigations
IEDB View	Vaccine Ontology

We also recorded the more complex logical constructs of the patterns in MBOP (Table 3). A pattern that utilizes one of these constructs cannot belong to the OWL 2 EL Profile, a less expressive, more computationally efficient fragment of OWL 2.

4 Discussion

Our results show a very modest use of patterns in biomedical ontologies in BioPortal. Of the 68 patterns that we studied (which we filtered from the initial list of 167), only 14 appeared among the almost 300 ontologies. Ontology developers may utilize patterns because of documentation, popularity, and support in development tools. For example, all ontologies using `Upper Level Ontology`, the most common pattern, instantiated the BFO, likely due to the OBO Foundry's popularity in the biomedical community and its emphasis on using an upper level ontology. The Protégé ontology development environment provides a quick method to instantiate `Closure`, `Value Partition` (DisjointUnionOf), and `Composite Property Chaining`. We believe that the easy accessibility of these patterns from this tool accounts for their use. While `Composite Property Chaining` is not explicit in Protégé, after creating property chains, composing them is trivial. `Normalization` is both a simple idea to follow (perhaps difficult in practice), and well explained by Rector et al. [20]. Finally,

Table 3. OWL 2 logical constructs in MBOP patterns.

	All Values From	Functional Property	Cardinality Restriction	Disjunction (Union)
Entity Feature Value		✔	✔	✔
Selector				✔
Normalization				
Upper Level Ontology Closure	✔			
Entity Quality	✔		✔	✔
Value Partition				✔
Entity Property Quality		✔		✔
Defined Class Description				
Interaction Role Interaction	✔		✔	✔
Sequence		✔		
Composite Property Chaining				
List		✔		
Adapted Structure, Entity, Part				✔
Nary Datatype				
Exception			✔	✔
Nary Relationship				

with only a few additions to the Relations Ontology (RO), an ontology that can be used along with BFO, the `Sequence` pattern from MBOP can be used.

We found only a subset of patterns in a subset of the BioPortal ontologies. Despite the suggested positive effects of ODPs, the relatively modest use of ODPs in BioPortal may be due to ontology age, domain specificity, minimal tooling support, lack of pattern portability and generality, and a tension between the high logical expressivity of certain patterns and a concurrent desire for minimally expressive biomedical ontologies to enable computationally tractable reasoning (because of their large size). With regard to ontology age, it is clear that older ontologies that began development before the introduction of ODPs may not have them, as restructuring an ontology to incorporate an ODP during maintenance may be impractical. Also, one might suggest that many patterns are not found simply because they are not related to the biomedical domain.

Of more relevance is the apparent lack of end-user tooling for instantiating ODPs. The XD Tools provide with such functionality as a plugin [6]. OPPL is also available as a Protégé plugin. However, a domain expert may not be familiar with ODPs, may not necessarily know how to use ODPs or OPPL, and may not even seek out the plugins. Concurrently, Protégé and other widely used ontology development tools do not have a general method to instantiate ODPs. However, for those patterns that are already available in a development environment (`Closure`, `Value Partition`), we do see their occurrence. In this study, we cannot easily link pattern usage and any particular ontology development environment. However, most biomedical ontology developers do use Protégé. Thus, we suggest that for ODP use to increase, end-user oriented ontology

development environments must include a way to instantiate various patterns, including those available in the ODP public libraries. Furthermore, such tools could note the use of patterns in the source file of the ontology.

We have noted the opposing tension in the expressivity in the patterns and biomedical ontologies. The OWL 2 EL profile in many ways was designed with large scale ontologies in mind. SNOMED CT [22, 4] served as its driving example. Table 3, which lists the constructs that each pattern uses, shows that only 25% of the MBOP patterns follow the OWL 2 EL profile. There is a dilemma: Patterns are designed to assist in reducing errors in large ontologies, but by using them, reasoning (and finding errors) becomes computationally intractable. This dilemma may explain the lack of ODP use by some ontologies, as they maintain something similar to EL expressivity.

Additionally, we found that BioPortal ontologies used more patterns from MBOP than from ODP-Wiki. MBOP has patterns oriented toward biomedical applications, explaining a portion of this bias. However, we found very few patterns from ODP-Wiki. Many patterns in ODP-Wiki either were domain specific, or, by definition, their instantiation required inclusion of a small ontology unit. While many ontologies might follow the intension of the content ODPs on ODP-Wiki, they did not import the required ontology to truly instantiate the pattern. Thus, more generalizable patterns, and a method to capture the intension of a content ODP might also increase ODP use.

4.1 Future Work

We consider the formalization of software-design patterns in a bottom–up fashion. Gamma and colleagues extracted recurring patterns from existing software, suggesting these patterns constituted best practice in solving various software development problems [10]. Contrary to this method, the development of ODPs in the Semantic Web community appears to be top–down, especially in the case of content ODPs. Instead, we propose to find ODPs in a bottom-up fashion, as with software design patterns, by finding recurring patterns in large corpora of ontologies, such as BioPortal.

5 Conclusions

Ontology Design Patterns provide a means to enhance ontology development by capturing best practice and reducing errors. As such, ODPs may be especially applicable to large-scale biomedical ontology development. As a starting point for a larger project, we first find the prevalence of ODPs in biomedical ontologies. To do so, using the available software for manipulating ODPs, we surveyed their use in BioPortal,a large repository of biomedical ontologies. We found only a small subset of patterns in use in a portion of the corpus, with `Upper Level Ontology` being the most common pattern. To increase future ODP use in biomedical ontologies, we highlight a need for end-user ontology development tools that include a way to instantiate ODPs and a need for consideration of the logical expressiveness of ODPs . Finally, we suggest a bottom-up approach to develop generalized ODPs for re-use.

Acknowledgments

This work was supported by the NIH Grants GM086587, HG004028, and LM007033. We thank Luigi Iannone for assistance with OPPL.

References

1. Aranguren, M.E., Antezana, E., Kuiper, M., Stevens, R.: Ontology design patterns for bio-ontologies: a case study on the cell cycle ontology. BMC bioinformatics 9(Suppl 5), S1–S1 (2008)
2. Aranguren, M.E., Antezana, E., Stevens, R.: Transforming the axiomisation of ontologies: The ontology pre-processor language. In: Proceedings of OWLED (2008)
3. Aranguren, M.E., Rector, A., Stevens, R., Antezana, E.: Applying ontology design patterns in bio-ontologies. Knowledge Engineering: Practice and Patterns pp. 7–16 (2008)
4. Baader, F., Brand, S., Lutz, C.: Pushing the EL envelope. In: Proc. of IJCAI 2005. pp. 364–369. Morgan-Kaufmann Publishers (2005)
5. Blomqvist, E.: Semi-automatic Ontology Construction based on Patterns. Ph.D. thesis, Linkoping University Institute of Technology (2009)
6. Blomqvist, E., Presutti, V., Daga, E., Gangemi, A.: Experimenting with extreme design. In: Proceedings of EKAW'10. pp. 120–134. Springer-Verlag, Berlin, Heidelberg (2010)
7. Bodenreider, O., Stevens, R.: Bio-ontologies: current trends and future directions. Briefings in bioinformatics 7(3), 256–274 (2006)
8. Ceusters, W., Smith, B., Goldberg, L.: A terminological and ontological analysis of the nci thesaurus. Methods of information in medicine 44(4), 498–507 (2005)
9. Daga, E., Presutti, V., Gangemi, A., Salvati, A.: http://ontologydesignpatterns. org [odp]
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Pearson Education (1995)
11. Gangemi, A., Presutti, V.: Ontology design patterns. Handbook on Ontologies pp. 221–243 (2009)
12. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., Schneider, L., Gómez-Pérez, A., Benjamins, V.: Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web, vol. 2473, pp. 223–233. Springer Berlin / Heidelberg (2002)
13. Grenon, P., Smith, B., Goldberg, L.: Biodynamic ontology: Applying BFO in the biomedical domain. Stud. Health Technol. Inform 102, 20–38 (2004)
14. Horridge, M., Bechhofer, S.: The OWL API: a Java API for working with OWL 2 ontologies. In: Proceedings of OWLED. vol. 529 (2009)
15. Iannone, L., Aranguren, M.E., Rector, A., Stevens, R.: Augmenting the expressivity of the ontology pre-processor language. In: Proceedings of OWLED (2008)
16. Iannone, L., Rector, A., Stevens, R.: Embedding knowledge patterns into OWL. The Semantic Web: Research and Applications pp. 218–232 (2009)
17. Niles, I., Pease, A.: Towards a standard upper ontology. In: Proceedings of the international conference on Formal Ontology in Information Systems - Volume 2001. pp. 2–9. FOIS '01, ACM, New York, NY, USA (2001)
18. Noy, N.F., Shah, N.H., Whetzel, P.L., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D.L., Storey, M.A., Chute, C.G., Musen, M.A.: Bioportal: ontologies and integrated data resources at the click of a mouse. Nucleic Acids Research 37, W170–W173 (05 2009)
19. Rector, A.L., Brandt, S., Schneider, T.: Getting the foot out of the pelvis: modeling problems affecting use of snomed ct hierarchies in practical applications. Journal of the American Medical Informatics Association 18(4), 432–440 (04 2011)
20. Rector, A.L.: Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In: Proceedings of the 2nd international conference on Knowledge capture. pp. 121–128. K-CAP '03, ACM, New York, NY, USA (2003)
21. Rubin, D.L., Shah, N.H., Noy, N.F.: Biomedical ontologies: a functional perspective. Briefings in Bioinformatics 9(1), 75–90 (2008)
22. Spackman, K.: An examination of OWL and the requirements of a large health care terminology. In: Proceedings of OWLED (2007)

A Pattern For Interrelated Numerical Properties

Jesper Zedlitz[1], Hagen Peters[2], and Norbert Luttenberger[2]

[1] German National Library of Economics (ZBW)
[2] Christian-Albrechts-Universität zu Kiel

Abstract. "A childs year of birth is always greater than the year of birth of its parents." – it is not easily possible to code this simple knowledge into a pure OWL ontology, i.e. without using any additional rule languages. Therefore it is not easy in OWL to detect semantic violations in this kind of statements. The two challenges are putting two orders ("greater" and "parent") into relation and representing integers as individuals allowing a reasoner to infer knowledge about the "greater" relation. In the first part of this contribution we show a pattern for putting two transitive and asymmetric orders into a relation, such that conflicting information results in an inconsistent ontology. In the second part we present a pattern for expressing integers using their binary code. Due to the special construction a reasoner can infer knowledge about the relation between all integers in the ontology. By combining the two patterns we are able to represent the initial statement in an ontology.

1 Introduction

The Web Ontology Language (OWL) makes a deliberate distinction between object properties of objects and data properties of objects. Data properties are designed to take a certain value from a range that is defined by the associated data type. However, the OWL specification does not go beyond sheer value assignment—no other operations are foreseen for the values of data properties. During the design phase of OWL 2, a number of authors therefore brought forward "wish lists" for different kind of operations on the values of data properties:

- Pan and Horrocks [1] propose the idea to enable calculations with data property values.
- In Use Case #10 of the W3C Working Draft [2] the value—not just the presence—of a data property is intended to be used for classification of individuals into classes.
- Grau et al.[3] list four kinds of operations to be provided for data property values. Among these is the requirement that it should be possible to express relations between values of data properties on different objects.

Although OWL's current version OWL 2 brought a number of enhancements to data type handling, the situation basically remained the same: the cited W3C Working Draft states that none of these wishes has been accepted for OWL 2.

Instead—in order to separate logical reasoning and handling of data values—the decision was taken to concentrate all data value handling inside the Semantic Web Rule Language (SWRL)[1] and its "built-ins". Integrated processing of OWL ontologies and SWRL rule sets accordingly requires software systems that comprise both a reasoner and an oracle for these built-ins. However, it obviously depends on the oracle's implementation how SWRL rules are evaluated (open vs. closed-world processing, data types). Furthermore the semantics of SWRL built-ins is outside the OWL ontology semantics.

In this paper, we present Logical Ontology Design Patterns (ODP) [4] for evaluating certain relations between values on different objects and for representing values from the range of integers as individuals in the ontology. Using these patterns semantic violations in data sets can be detected during consistency checks.

This paper is organized as follows: In section 2 we describe the class of problems we want to address and discuss other approaches to the problem. For clarity we split our solution into three separate parts: Section 3 presents an ODP for putting two orders into a relation. Section 4 presents our ODP for representing integers. In section 5 we introduce a third ODP that combines both previous ODP and prove the correctness of our approach. Section 6 concludes and points out open questions. We give examples written in OWL 2 functional-style syntax [5] where appropriate.

2 Problem Description & Approach

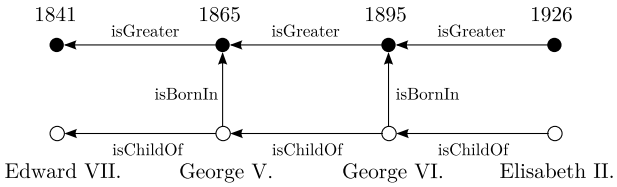


Fig. 1. Example of the problem we address.

As an example for the kind of problems we want to address let us take a look at this situation: A person P_1 has an *isChildOf* relation to another person P_2. Each of the persons has a relation *isBornIn* to its year of birth. This information is coded in an OWL 2 ontology. The ontology shall only be consistent if the year of birth of P_1 is greater than the year of birth of P_2: "A child's year of birth is always greater than the year of birth of its parents." Figure 1 illustrates this example.

There are several ways known to model the above mentioned situation and to express the relation of birth years of parents and their children. One could

[1] http://www.w3.org/Submission/SWRL/

use rule languages like SWRL to express this knowledge. But there are only a few reasoners that (fully) support rules at all and many different rule languages are used. Furthermore, as mentioned in the introduction, at least parts of the semantics of the rule languages is outside of the semantics of OWL. Therefore the evaluation of these parts is rather asking an oracle (i.e. using the specific implementation) than OWL reasoning. For example, the specification of data types in SWRL allows for different implementations in terms of precision of decimals.[2] Hence two different reasoners, both supporting the same parts of OWL and being compliant to the SWRL specification could evaluate the same ontology differently.

For this reason we focus on approaches that only use pure OWL evaluation. There are two main approaches to express relations between natural numbers (in the following "integers" w.r.t. common data type definitions) as in our example: representing integers as literals and representing integers as individuals.

Representing integers as literals The common way to model the above scenario with integers represented as literals is to use data properties for the year of birth and a combination of restrictions on object and data types to detect semantic violations. Listing 1 shows a restriction for the individual named "GeorgeV" requiring his father's birth year to be before 1865.

```
1  Declaration( NamedIndividual( :EdwardVII ) )
2  Declaration( NamedIndividual( :GeorgeV ) )
3  ClassAssertion(
4     ObjectAllValuesFrom( :isChildOf
5        DataAllValuesFrom( :isBornIn
6           DatatypeRestriction( xsd:integer xsd:maxExclusive
7              "1865"^^xsd:integer )
8        )
9     )
10    :GeorgeV
11 )
12
13 DataPropertyAssertion( :isBornIn :GeorgeV "1865"^^xsd:integer )
14 DataPropertyAssertion( :isBornIn :EdwardVII "1841"^^xsd:integer )
15 ObjectPropertyAssertion( :isChildOf :GeorgeV :EdwardVII )
```

Listing 1. Using data properties and data type restrictions

However, although this approach seems more or less obvious it suffers from two major disadvantages:

1. No **general** statements about birth years of parents and children are made, but for each child the maximum birth year of its parents has to be specified. That requires additional axioms for each individual that belongs to the ordered set.
2. Now there is a restriction on the parent's birth year but there is **no** formal correspondence between this restriction (line 7) and George's birth (line 13). That means, the ontology could also be consistent if one restricts the parent's

[2] *"All minimally conforming processors must support decimal numbers with a minimum of 18 decimal digits"* from http://www.w3.org/TR/xmlschema-2/ section 3.2.3

birth year to a maximum value of 1865 (line 7) while (in line 13) George's birth year is (e.g. by mistake) set to 1800, which is obviously not intended.

In summary, with this approach we are not able to express a general statement about the relation of the years of birth of parents and their children but have to express that knowledge for each of the parent-child relations explicitly (1). Furthermore, this approach still allows for semantic violations (2).

Representing numbers as individuals Other authors propose the use of resources (i.e. OWL individuals) rather than literals for the representation of numbers. [6] shows several advantages of this approach. Most interesting for our problem is the possibility to reason about relations between these number individuals and other individuals of the ontology. However, in this approach the name (IRI) of an individual is used to encode some knowledge about the resource. Since names of individuals are meaningless character sequences in terms of formal reasoning, it is a priori not possible to use the knowledge encoded in the individual's names during the reasoning process.

Another approach for representing an integer n is to specify the predecessor (and/or successor) of n. Using this approach implies, that if a particular integer n is needed in the ontology all $n-1$ predecessors must be part of the ontology, too. Thus the representation of an integer depends on other individuals that are (or aren't) contained in the ontology. This obviously implies that adding an integer representing individual is a non-trivial task and requires full knowledge of the ontology.

It might also be possible to represent an integer n by using an individual having n properties and adding appropiate cardinality constraining axioms. In both approaches, using predecessors as well as n properties, the number of axioms needed to represent a single integer scales linearly with the value of the integer.

Thus these approaches require large maintenance effort (adding statements about numbers not actually used, changing the definition of previously defined numbers, etc.) and knowledge about already existing integers in the ontology. Furthermore the linear dependency between the value of an integer and the number of axioms needed in the ontology makes these approaches impractical for many scenarios.

Our approach In our approach we want to use individuals to represent integers. Our goal is to find a pattern where

1. the number of axioms needed for representing a single integer depends only logarithmically on the value of that integer (like the usual binary or decimal representation),
2. the representation of an integer is independent of whether or not other integers already exist in the ontology, and
3. it is possible to reason about integers based on their representation.

Once we found that pattern we'll be able to detect "direct" semantic violation of the order of integers, e.g. having two integers 3 and 4 and an the explicit statement like "3 is greater than 4".

However, that is not enough for our initial problem, i.e. detecting semantic violations on birth years of parents and their children. In this problem we want to detect "indirect" semantic violations, e.g. person P_1 is the child of person P_2, **but** P_2 is born before P_1. In other words, we have to detect semantic violations between two orders (the order given by birth years *isGreater* and the order given by the child relation *isChildOf*) that are connected by another property (*isBornIn*).

In the next section we show a pattern putting two orders into relation. Section 4 describes our pattern for integer representation.

3 Comparing Two Orders

The first part of our solution is a pattern for putting two arbitrary orders into relation. To outline that this pattern is not restricted to numerical values we use another example here. In Fig. 2 the individuals depicted with black-filled circles represent (in this case non-numeric) time data. The *isYoungerThan* object property establishes an order on these individuals. Analogously the individuals depicted with white-filled circles (in Fig. 2: tools) were put into an order using the *isSuccessorOf* property. The *isToolOfThe* object property connects individuals of the one with individuals of the other order, i.e. tools with ages.

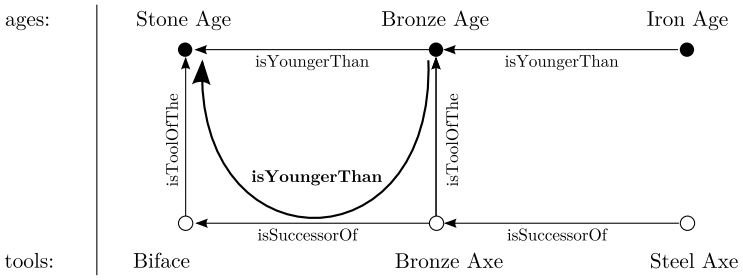


Fig. 2. Comparing two ordered sets of individuals.

The ontology should be inconsistent if two individuals in one order are connected to two individuals with inverse order. The trick is to use one of the order-building properties to infer knowledge about the other one. This can be achieved by using a chain of object properties:

$$isToolOfThe^{-} \circ isSuccessorOf \circ isToolOfThe \rightarrow isYoungerThan$$

To compare not only neighboring individuals in each order it is necessary to declare transitivity for (at least) one of the relations. In our example this would

be:

$$isYoungerThan(x, y) \wedge isYoungerThan(y, z) \rightarrow isYoungerThan(x, z)$$

3.1 Asymmetry

Furthermore, if contradicting information should result in inconsistency it is necessary that the following holds:

$$\forall x, y : isYoungerThan(x, y) \implies \neg isYoungerThan(y, x)$$

This could be achieved by marking the $isYoungerThan$ object property asymmetric. Unfortunately this is not possible in OWL 2, because to guarantee decidability[5, sec. 11.2] an object property must not be transitive and asymmetric at the same time.

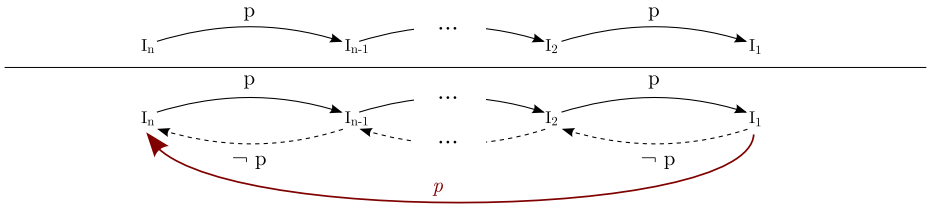


Fig. 3. (Negative) Object Property Assertions.

However, for the known individuals in the ontology it is possible to assure asymmetry by inserting negative object property assertions. To ensure that $\neg p(I_a, I_b)$ holds for arbitrary $b > a$, surprisingly only $n - 1$ negative object property assertions are necessary for n ordered individuals: The upper half of Fig. 3 shows the situation where object properties (depicted by solid arrows) are used to represent an order on individuals $I_1 \dots I_n$. To ensure asymmetry on $I_1 \dots I_n$ it is sufficient to only insert negative object property assertions between neighboring individuals (depicted by dashed arrows in the lower half of Fig. 3):

Assume the assertion $p(I_1, I_n)$ is part of the ontology sketched above (Fig. 3). Then the following inferences can be made:

$$p(I_1, I_n) \xrightarrow{p(I_n, I_{n-1})} p(I_1, I_{n-1}) \xrightarrow{p(I_{n-1}, I_{n-2})} \dots \xrightarrow{p(I_3, I_2)} p(I_1, I_2)$$

This is a contradiction to $\neg p(I_1, I_2)$. □

Thus for every x, y with $isYoungerThan(x, y)$ holds $\neg isYoungerThan(y, x)$, stated either explicitly or implicitly.

3.2 Summary I

We have now defined a pattern that enables us to put two orders defined on two sets of individuals—not necessarily integers—into relation. If one of the orders

contains individuals representing integers and the relation is a natural ordering on integers this technique solves part of our initial problem. It is just necessary to list the needed integers in the ontology instead of all integers in between. If n integers are used, $2(n-1)$ object properties are required to model the order in the ontology. A semantic violation in the specified ordering makes the ontology inconsistent.

The main disadvantage of this pattern is the fact that an individual has no formal relation to the value it is intended to represent (except for the label which is not part of the knowledge included in reasoning process). If one needs to insert an individual representing a specific value (lets say the "copper age" in Fig. 2) this is not possible without additional knowledge about the already existing individuals (e.g. the individual labeled "bronze age" is younger than "copper age"). Without additional knowledge it is only possible to insert an individual that is known to represent a new minimal or maximal value.

4 Integer Representation

In the second part of our solution we use a pattern that constructs classes containing information about the relation of integers in the ontology. We represent an integer in the binary numeral system with a predefined bit-length. Figure 4 shows every four-digit integer as a leaf of a binary tree (the gray shaded areas can be ignored for now). The binary representation of an integer is given by the path from the root to the leaf. The natural "greater than" order is given by the order the leaves appear in a depth-first tree traversal.

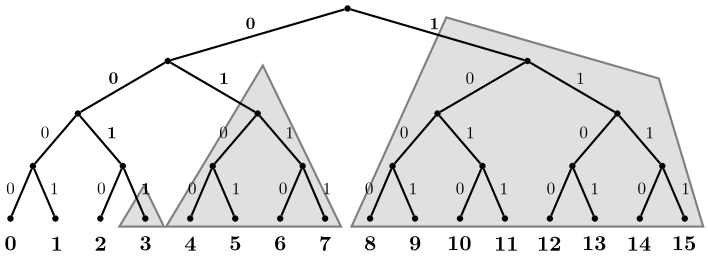


Fig. 4. Binary tree.

Let I_n denote an individual that represents the integer n in the ontology.[3] Let $C_{i.0}$ denote the class of integers with the i^{th} bit being zero. Let $C_{i.1}$ denote the class of integers with the i^{th} bit being one. (The least significant bit is bit zero.)

[3] We expect that there could be more than one individual in the ontology representing a given integer.

We put each required integer into its respective classes, e.g. for the individual "two".

$$I_2 \in C_{3.0}, I_2 \in C_{2.0}, I_2 \in C_{1.1}, \text{and } I_2 \in C_{0.0}.$$

```
ClassAssertion( :3.0 :Two)
ClassAssertion( :2.0 :Two)
ClassAssertion( :1.1 :Two)
ClassAssertion( :0.0 :Two)
```

Listing 2. Representing the 4-bit integer 2

Note that "two" is just an arbitrary label for an individual representing integer 2. It would not make any difference to label the individual e.g. "MyPersonalTwo".

Now we can define a class that contains all individuals that represent the same integer by an intersection of the classes of the binary digits, e.g. for the integer 2:

$$C_{equal.2} = C_{3.0} \cap C_{2.0} \cap C_{1.1} \cap C_{0.0}$$

```
SubClassOf(
    ObjectIntersectionOf(:3.0 :2.0 :1.1 :0.0 )
    :Equal2 )
```

Listing 3. Class containing all individuals representing integer 2

It is easy to see that two individuals representing the same integer n in the ontology belong to the same class $C_{equal.n}$. We note this observation as

Lemma 1 *Let I_a be an individual in the ontology representing the integer a and I_b be an individual representing the integer b. Let $a = b$. Let the ontology contain axioms following the construction rules mentioned above. Then $C_{equal.a}(I_b)$ is also entailed by the ontology.*

The class $C_{greater.n}$ (which contains all integers greater than n) are unions of sub-trees of our binary tree (see Fig. 4): According to the construction rules of the tree it holds for every sub-tree that all leafs that are descendants of the right child node ("1") are greater than every leaf that is a descendant of the left child node ("0").

Thus whenever the path representing an integer n follows a 0-edge all leafs that can be reached via the corresponding 1-edge must be included into $C_{greater.n}$. The set of leafs that can be reached via the 1-edge can be easily written as an intersection of binary classes.

As an example consider the construction of $C_{greater.2}$. The elements of the union can clearly be seen in Fig 4.

$$C_{greater.2} \subseteq C_{3.1} \cup (C_{3.0} \cap C_{2.1}) \cup (C_{3.0} \cap C_{2.0} \cap C_{1.1} \cap C_{0.1})$$

```
SubClassOf(
    ObjectIntersectionOf( :3.1 )
    :GreaterThan2 )
SubClassOf(
    ObjectIntersectionOf( :3.0 :2.1 )
```

```
    :GreaterThan2 )
SubClassOf(
    ObjectIntersectionOf( :3.0 :2.0 :1.1 :0.1 )
    :GreaterThan2 )
```

Listing 4. Class containing all individuals representing integers greater than 2

For every integer n used in our ontology we have to declare that any integer that is equal n must not be greater than n:

$$C_{equal.n} \cap C_{greater.n} = \emptyset$$

```
DisjointClasses(   :GreaterThan3 :Equal3 )
```

Listing 5. Definition of strictly greater

Following these construction rules knowledge about the relation of integer representing individuals can be inferred: An individual representing an integer greater than n belongs to the class $C_{greater.n}$. We note this observation as

Lemma 2 *Let I_a and I_b be two individuals in the ontology representing integers a and b (with $b > a$). Let the ontology contain axioms following the construction rules mentioned above. Then $C_{greater.a}(I_b)$ is also entailed by the ontology.*

Summary II

In contrast to the restriction of the first pattern of our solution the second pattern does not require any knowledge about the existing integers in the ontology when adding an individual representing an integer. It is only necessary to follow the named construction rules. Individuals representing integers are automatically put into relation with each other. There is only a linear relation in the number of assertions needed to represent an integer and the number of bits used for the binary representation. The setting of a bit length specifies a maximal number. However, using a defined bit length is very common for computer systems.

The pattern introduced above establishes only relations between individuals and classes not between individuals and object properties which are needed for our initial problem. We address this "gluing problem" in the next section.

5 Putting It Together

Now we can use the information from the classes as constructed in section 4 to establish an order on the integer individuals. The main problem is to create a link between the *greater* object property and classes. Our solution has been inspired by [7]. Formally written we need:

$$\begin{aligned} C_{greater.n} = &\{x | \exists y : greater(x, y) \wedge C_{equal.n}(y)\} \\ &\cup \{x | \exists y : greater(x, y) \wedge C_{greater.n}(y)\} \end{aligned}$$

It is not possible to state this linking of object properties and classes directly in OWL 2. However, by carefully using the *EquivalentClasses* class axiom and the class expressions *ObjectUnionOf* and *ObjectSomeValuesFrom* it is possible to make an equivalent statement. In OWL 2 functional syntax this axiom has to be added:

```
EquivalentClasses(
   :GreaterThanY
   ObjectUnionOf(
      ObjectSomeValuesFrom( :greater :EqualY )
      ObjectSomeValuesFrom( :greater :GreaterThanY ) ) )
```

Listing 6. Connection between the *greater* object property and classes

If this axiom and the class definitions shown above are included in the knowledge base, the existence of a *greater* property between two individuals classifies the individuals into the according "GreaterThan" classes. This knowledge about the *greater* relation can be used in combination with the first part of our approach. For the generic example sketched in Fig. 5 the following axioms would be part of the corresponding ontology:

```
Declaration( Class( :Number ) )
Declaration( ObjectProperty( :s ) )

Declaration( ObjectProperty( :r ) )
ObjectPropertyRange( :r :Number )

SubObjectPropertyOf( ObjectPropertyChain( ObjectInverseOf(:r) :s :r )
                     :greater )
```

Listing 7. Infer *greater* property

All integer individuals which are connected by the above mentioned property chain are put into a greater relation. Based on this relation the integer individuals are classified into the "GreaterThan" classes. Thereby individuals and object properties are "glued".

As shown in section 4 the integer individuals are already classified into "GreaterThan" classes based on their construction. Thus a semantic violation regarding the order of individuals or the connection to the individuals representing integers leads to contradicting classifications. In combination with the disjoint classes axioms this leads to an inconsistency in the ontology. In the next section we show this formally.

5.1 Proof

Fig. 5. Sketch of the ontology used in the proof. The "order" is given by object property s, the individuals are X and Y and the integers are I_a and I_b. X and Y are connected to the integers via property r.

To show the correctness of our approach we consider an ontology constructed following the rules above and sketched in Fig 5. It consists of two individuals X and Y that are connected via an object property $s(X, Y)$. The two individuals I_a and I_b represent two integers a and b. X and Y are connected with these integers representing individuals via an object property r.

```
DisjointClasses( :GreaterThanA :EqualA )
EquivalentClasses(
   :GreaterThanA
   ObjectUnionOf(
      ObjectSomeValuesFrom( :greater :EqualA ) )
      ObjectSomeValuesFrom( :greater :GreaterThanA ) ) )

DisjointClasses( :GreaterThanB :EqualB )
EquivalentClasses(
   :GreaterThanB
   ObjectUnionOf(
      ObjectSomeValuesFrom( :greater :EqualB ) )
      ObjectSomeValuesFrom( :greater :GreaterThanB ) ) )
```

Listing 8. Application of listings 5 and 7 for the example of Fig. 5.

Listing 8 can be written formally as:

$$C_{greater.a} \cap C_{equal.a} = \emptyset \tag{1}$$

$$C_{greater.a} = \{x | \exists y : greater(x, y) \wedge C_{equal.a}(y)\} \tag{2}$$

$$\cup \{x | \exists y : greater(x, y) \wedge C_{greater.a}(y)\} \tag{3}$$

$$C_{greater.b} \cap C_{equal.b} = \emptyset \tag{4}$$

$$C_{greater.b} = \{x | \exists y : greater(x, y) \wedge C_{equal.b}(y)\} \tag{5}$$

$$\cup \{x | \exists y : greater(x, y) \wedge C_{greater.b}(y)\} \tag{6}$$

Using the property chain (described in section 3) axiom (cf. listing 7) we can infer:

$$r(X, I_a)^- \wedge s(X, Y) \wedge r(Y, I_b) \Rightarrow greater(I_a, I_b) \tag{7}$$

By construction $C_{equal.a}(I_a)$ and $C_{equal.b}(I_b)$ always hold. According to the ontology depicted in Fig. 5 in every case $greater(I_a, I_b)$ is inferred (formula 7). We have to distinguish three cases:

1. $a > b$: $C_{greater.b}(I_a)$ (lemma 1)

$$greater(I_a, I_b) \wedge C_{equal.b}(I_b) \overset{(5)}{\Longrightarrow} C_{greater.b}(I_a)$$

2. $a = b$: $C_{equal.a}(I_b)$ (lemma 1)

$$greater(I_a, I_b) \wedge C_{equal.a}(I_b) \overset{(2)}{\Longrightarrow} C_{greater.a}(I_a)$$

   This is a contradiction to (1) with $C_{equal.a}(I_a)$.

3. $a < b$: $C_{greater.a}(I_b)$ (lemma 2)

$$greater(I_a, I_b) \wedge C_{greater.a}(I_b) \overset{(3)}{\Longrightarrow} C_{greater.a}(I_a)$$

   This is a contradiction to (1) with $C_{equal.a}(I_a)$.

Thus, the semantic violations in cases (2) and (3) result in an inconsistent ontology.

6 Conclusion

In this contribution we present a pattern for putting two orders into relation and a second pattern for representing integers as individuals of an ontology. Following the simple construction rules allows for inference of relations between these individuals. The combination of both patterns can be used to detect contradicting information regarding the two orders. If the ontology contains contradicting information the ontology will become inconsistent. The correctness of our combined pattern is shown in section 5.1.

Our patterns might be beneficial for verification purposes. When testing the performance of the consistency check using test ontologies it turns out that our pattern is not adequate for representing huge amounts of different integers. However, on the one hand in many use cases the time complexity is not worse than a solution that uses only data properties and on the other hand many use-cases do not require many different integers, like the one sketched in Fig. 2.

The authors are aware of the fact that it is possible—if you do not follow the construction rules for integer individuals and relations between them—to create an ontology that is consistent although the represented information contains a semantic violation—for example if statements in listing 2 do not match statements in listings 3 and 4. A solution would be to infer the relations between integer-representing individuals directly from the representation of these integers, but there are some indications that this is not possible. However, it is an open problem to prove this.

Another interesting question would be to investigate if it is possible to model the relation between more than two integers using OWL 2. That could be used to express and check relations that include arithmetic operations. If it should be possible, variants of the initial statement could also be expressed: "A child's year of birth is always at least 10 years greater than the year of birth of its parents."

References

1. Pan, J.Z., Horrocks, I.: Web Ontology Reasoning with Datatype Groups. In: Proc. of the 2nd International Semantic Web Conference. (2003) 47–63
2. Golbreich, C., Wallace, E., Patel-Schneider, P.: OWL 2 Web Ontology Language: New Features and Rationale. W3C working draft (2009)
3. Cuenca Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: OWL 2: The next step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web **6**(4) (2008) 309–322
4. Gangemi, A., Presutti, V.: Ontology design patterns. Handbook on Ontologies (2009) 221–243
5. Motik et al. (eds.): OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Recommendation (2009)
6. Champin, P.: Representing data as resources in RDF and OWL. Proc. of the 1st Workshop on Emerging Research Opportunities for Web Data Management (2007)
7. Tsarkov, D., Sattler, U., Stevens, M., Stevens, R.: A solution for the Man-Man problem in the Family History Knowledge Base. In: Proc. of the 5th International Workshop on OWL: Experiences and Directions. (2009) 23–24

Linguistic Patterns for Information Extraction in OntoCmaps

Amal Zouaq[1,2], Dragan Gasevic[2,3], Marek Hatala[3]

[1]Royal Military College of Canada, CP 17000, Succ. Forces, Kingston, ON Canada K7K 7B4
[2]Athabasca University, 1 University Drive, Athabasca, AB T9S 3A3
[3]Simon Fraser University, 250-102nd Avenue, Surrey, BC Canada V3T 0A3
Amal.zouaq@rmc.ca, dgasevic@acm.org, mhatala@sfu.ca

Abstract. Linguistic patterns have proven their importance for the knowledge engineering field especially with the ever-increasing amount of available data. This is especially true for the Semantic Web, which relies on a formalization of knowledge into triples and linked data. This paper presents a number of syntactic patterns, based on dependency grammars, which output triples useful for the ontology learning task. Our experimental results show that these patterns are a good starting base for text mining initiatives in general and ontology learning in particular.

Keywords: Linguistic patterns, dependency grammars, triples, knowledge extraction, ontology learning.

1 Introduction

With the development of Semantic Web technologies and the increased number of initiatives relative to the Web of data, there is a need to create reusable and high quality ontologies. For this purpose, ontology design patterns (ODP) have been modeled and span various aspects of ontological design such as architectural ODPs, Content ODPs and Reengineering ODPs [1]. These patterns enable the definition of formal methodologies for ontology creation and maintenance. Of particular interest to the knowledge engineering community are Lexico-Syntactic Patterns. In fact, with the availability of large unstructured knowledge resources such as Wikipedia, it becomes crucial to be able to extract ontological content using scalable (semi)automatic knowledge extraction techniques. In this work, we consider that lexico-syntactic patterns are syntactic structures that trigger the extraction of chunks of information. Obviously, these patterns cannot be used in isolation and they necessitate various filtering mechanisms [6] to identify *ontological knowledge* from this extracted *textual information*. However, by constituting Lexico-syntactic ODP catalogs, it is likely that these ODPs will be used and reused as the first building block of ontology learning initiatives. So far, there are some ODPs repositories [2] but their number remains modest.

Trying to address this issue from a knowledge extraction perspective, this paper introduces the main patterns that are used by OntoCmaps, an ontology learning tool [6]. OntoCmaps exploits a pattern knowledge base that is domain independent. OntoCmaps lexico-syntactic ODPs are based on a dependency grammar formalism [5] that is well-suited for knowledge extraction. This paper describes the various ODPs that are used to extract multi-word expressions, hierarchical relationships and conceptual relationships that can be later promoted as domain knowledge and converted in OWL[1] format. The paper details each class of patterns, and presents the accuracy of each pattern prior to any filtering. Our results show that filtering techniques should be used in a separate sieve on top of the ODPs to improve the accuracy of the extraction.

2 Related Work

ODP are quite recent design patterns whose objective is to create modeling solutions to well-identified problems in ontology engineering and thus promote good design. In this paper, we are mostly interested in one subclass of ODPs: Lexico-syntactic patterns (LSPs). In particular LSPs are meant to identify which patterns in texts correspond to logical constructs of the OWL language. This type of patterns is essential from a semi-automatic ontology engineering point of view. In fact, as soon as the domain becomes a real-world problem, it starts to be difficult and very expensive to manually design an ontology.

In general LSPs are a widely used method in text mining and can be traced back to the work of [4] for hyponymy extraction. There have been several attempts to use LSPs for ontology learning [2, 3, 6, 7], relation extraction [7] or for axiom extraction [3]. Among the most similar works to ours are SPRAT [7] and [2] which propose various patterns for ontology learning and population. One of the peculiarities of our approach is that we designed purely syntactic patterns that examine a bigger number of linguistic constructs (e.g. relative clause modifiers, adjectival complements, copula, etc.) than what is available in the state of the art to extract information from text. Moreover, our patterns are based solely on a dependency grammar combined with parts-of-speech tagging. We used a similar approach in a previous work [8] on semantic analysis but the aim was not the extraction of triples for ontology learning and the patterns themselves were not structured and conceived in the same way. Finally, there are also various LSPs identified on the ODP portal[2] but to the best of our knowledge, the majority of the patterns in this paper are new with respect to the listed LSPs. Overall, 29 over the 31 patterns presented in this paper are not listed on the ODP portal. There is one common pattern for object property extraction (*nsubj-dobj*) which is already widely used in the information extraction field and one common pattern for hierarchical relations extraction (*nsubj-cop*). Finally, in one case, there is a similarity between one LSP used to extract sub-classes/super-classes relationships and one of our patterns for hierarchical relationship extraction (with the use of the expression *including* instead of *include*). In any case, one major difference is the use of dependency relations in OntoCmaps.

[1] http://www.w3.org/TR/owl-features/

[2] http://ontologydesignpatterns.org/wiki/Category:ProposedLexicoSyntacticOP

3 Lexico-Syntactic Patterns in OntoCmaps

3.1 OntoCmaps

OntoCmaps is an ontology learning tool that takes unstructured texts about a domain of interest as input [6]. OntoCmaps is essentially based on two main stages: a knowledge extraction step which relies on syntactic patterns to extract candidate triples from texts, and a knowledge filtering step which acts as a sieve to identify relevant triples among these candidates. Since this paper focuses on the knowledge extraction part, this section presents the formalisms and tools used during the extraction stage.

As aforementioned, OntoCmaps patterns are mainly syntactic patterns which use a dependency grammar formalism and part-of-speech tagging [9]. The dependency analysis is obtained through the Stanford Parser [5] which defines a grammatical relations hierarchy and outputs dependencies (we use the collapsed dependencies). A dependency parse represents a set of grammatical relations (from this hierarchy) that link each pair of related words in a sentence. Several examples of dependency parses are provided in the following sections.

3.2 Syntactic Patterns

Patterns define specific syntactic configurations that link variables, constrained by given parts-of-speech, using grammatical dependencies. Parts of speech constraints allow filling in the variables with the right types of grammatical categories and therefore are essential for the accuracy of the extraction [8]. Parts of speech are defined in the Penn Treebank II[3].

Some patterns might overlap and are organized into a pattern hierarchy to trigger the more detailed patterns first. When a parent pattern is instantiated, all its children are disqualified for the current sentence, to avoid the extraction of meaningless fragments. Patterns are interpreted (using Java methods) to extract triples that can represent candidate domain ontological relationships. Triples also let OntoCmaps identify potentially relevant domain terms (i.e. content words).

The transformation rules (which can be considered as Transformation ODPs) focus on triples to enable mappings with the OWL-DL language. OWL DL is much more limited than natural language. Consequently, various syntactic configurations do not have any equivalent in OWL-DL. For example, verbs tense cannot be represented. There are general conventions that are followed in OntoCmaps for generating possible mappings: 1) Nouns and combinations of nouns, adjectives and adverbial modifiers are converted into potential candidate classes; 2) Proper nouns are converted into named entities (potential instances); 3) Comparative adjectives potentially map to an OWL Object Property when domain and range are classes; 4) Transitive verbs map to potential OWL Object Properties when domain and range are classes. They can also map to data types properties if the range is not considered as a class; 5) Negation on a

[3] http://www.cis.upenn.edu/~treebank/

verb between two identified classes maps to the OWL complement construct; 7) Verb tenses, modals and particles are all aggregated in the label of a potential OWL object property (verb); 8) The noun following a possessive pronoun is translated into and OWL Object Property or a data type property; 9) Determiners, quantifiers, comparative and superlative adverbs are ignored in OntoCmaps at this point.
There is no predefined meaning assigned to any of the extracted terms and relationships. Terms and relationships labels might have various morphological forms but are all related to their root lemma. Therefore, various morphological structures with the same root all relate to the same relation or term. Semantics is left underspecified or more specifically specified by the domain context, since OntoCmaps takes a domain corpus as input. If there are triples related to the term "bank", then whether it is the financial institution or the side of a river will be determined by the input corpus and by the other extracted relationships. The following sections details the patterns used by OntoCmaps. A pattern is represented using the following convention:

Grammatical relation (Head-Index / POS, Dependent-Index/POS) → Transformation

- Grammatical relation represents a dependency relation;
- Head and Dependent are variable names;
- POS represents a part-of-speech. Note that we use the generic part-of-speech NN for all the noun parts-of-speech (NN, NNS) and the generic part-of-speech VB for all the verbal parts-of-speech (VB, VBD, VBG, VBN, VBP and VBZ);
- Index represents the position of the word in the sentence;
- Transformation describes the resulting expression when this pattern is instantiated.

3.3 Expression Extraction

Simple and multi-word expressions (MWE) are considered candidate domain terms if they occur in lexico-syntactic ODPs for hierarchical and conceptual relationship extraction. The first step in OntoCmaps consists of aggregating MWE to generate a new dependency graph composed of MWE linked by grammatical relations (Table 1). The most common MWE are obtained through the patterns (1), (2) and (3).

Table 1. Multi-word expressions patterns

Pattern	Example
`nn(X/NN, Y/NN)` → `Y_X (1)`	nn(Systems-3/NN, Computer-1/NN), nn(Systems-3/NN, Operating-2//NN) →Computer operating systems
`amod(X/NN, Y/JJ)→Y_Z (2)`	amod(intelligence-2/NN, Artificial-1/JJ) → Artificial Intelligence
`nn(X,Y), amod(X,Z)` → `Z_Y_X (3)`	amod(systems-3/NN, Intelligent-1/JJ), nn(systems-3/NN, computing-2/NN)→Intelligent computing systems Note that Pattern (3) is a combination of (1) and (2).

`advmod(X/VBN, Y/RB), dobj(X/VBN, Z/NN) → Y_X_Z (4)`	advmod(modified-2/VB, Experimentally-1/RB), dobj(modified-2/VB, cell-3/NN)→ Experimentally modified cell
`prep_of/IN (X/NNP, Y/NNP)→ X_of_Y(5)`	prep_of/IN (University-2//NNP, Toronto-4/ NNP)→ University of Toronto
`prep_of /IN(X/NN, Y/NN)→ X_of_Y (6)`	prep_of/IN(page-2/NN, book-5/NN)→ Page of book Note that another possible transformation could be Y X (e.g. book page).

We also designed two patterns for multi-word expressions containing the preposition *of* (5) (6). Pattern (5) extracts a named entity and is useful for ontology population (rather than learning). The type of multi-word expressions in (6) can be tricky, as the Y part can represent the domain concept and the X part might be only an attribute (e.g. the color of the car) or a part (the wheel of the car). However, if this MWE is not important for the domain, it is likely that it will be sieved during the filtering stage.

3.4 Relationship Extraction

Relationship extraction in OntoCmaps refers to hierarchical relationships extraction (aka taxonomy or hyponymy) and conceptual relationships extraction (OWL Object Properties). Relationship extraction is run after few other operations, mainly the aggregation of multi-words expressions, the distributive interpretation of conjunctions and the removal of function words such as determiners and quantifiers. These prior operations produce a modified dependency graph used as input for relationships extraction.

Hierarchical Relationship Extraction.

There have been many works [2, 4, 10] in hierarchical relationships extraction using patterns. In OntoCmaps, hierarchical relationships are mapped to subclasses in OWL-DL.

OntoCmaps reuses some of Hearst's patterns (patterns (7) (8) in Table 2) using the dependency grammar formalism, parts-of speech, and transformation rules. We also create a hierarchical relationship from the multi-word expression pattern (3) (in Table 1) which involves a noun compound modifier and an adjectival modifier (pattern 9, Table 2) and from the multi-word expression pattern 4 (Table 1) thus obtaining pattern (10) (Table 2). Finally, we designed one pattern based on the copula (Pattern (11), Table 2).

Table 2. Hierarchical Relations Patterns

Pattern	Example
`prep_such_as/IN(X/NNS,`	Animals such as monkeys and apes prep_such_as/IN(animals-1/NNS, monkeys-

Y/NNS)→is-a(Y, X) (7)	4/NNS), prep_such_as/IN(animals-1/NNS, apes-6/NNS) → *is_a (monkeys, animals); is-a(apes, animals)*
prep_including/IN(X/NNS, Y/NNS)→is-a(Y, X) (8)	All buildings, including houses and castles… prep_including/IN(buildings-2/NNS, houses-5/NNS*)* → *is-a(houses, buildings); is-a(castles, buildings)*
nn(X,Y), amod(Y,Z)→ is-a (Y, X)(9)	Intelligent computing systems… amod(systems-3, Intelligent-1), nn(systems-3, computing-2*)* → *is-a (Intelligent computing systems, computing systems)* → *is-a(Computing systems, systems)*
advmod(X/VBN, Y/RB), dobj(X/VBN, Z/NN) → is-a(Y_X_Z, Z)(10)	Genetically modified food advmod(modified-2, Genetically-1), dobj(modified-2, food-3)→ *is-a(Genetically modified food, food)*
nsubj(X/NNS, Y/NNS), cop(X/NNS, Z/VBP)→ is-a(Y,X)(11)	Pinguins are birds nsubj(birds-3/NNS, Pinguins-1/NNS), cop(birds-3/NNS, are-2/VBP*)* → *is-a(Pinguins, Birds)*

Conceptual Relationships Extraction.

Conceptual relationships refer to OWL Object Properties with a domain and range. These relationships are among the most difficult to extract. We propose dependency-based patterns that are divided into four main categories: main clauses (containing a nominal subject nsubj), passive clauses (containing a passive nominal subject), relative clauses (containing a relative clause modifier) and finally other clauses which group certain constructs not belonging to the other categories.

Main clauses.

Main clauses are organized around the main verb of the sentence. Pattern (12) has been already referenced in the ODP portal. Pattern (13) enriches Pattern (12) with a preposition attached to the main verb and allows the creation of two triples. Patterns (16-18) use the xcomp relationship (which indicates a clausal complement with an external subject) to create a relationship between the main subject of the sentence (nsubj) and its direct object (Pattern (16) and (18)) or its related agent (Pattern (17)). Finally, Pattern (14) and (15) create a relationship between the nominal subject and the object of the preposition.

Table 3. Main clause patterns for conceptual relationship extraction

Pattern	Example
nsubj(X/VB, Y/NN),	Content packaging can define content organizations.

dobj(X/VB, Z/NN)→X(Y,Z) (12)	→ *can define(content packaging, content organizations)*
nsubj(X/VB, Y/NN), dobj(X/VB, Z/NN), prep_K(X/VB, A/NN)→ X_Z_K(Y, A), X(Y,Z)(13)	AICC has submitted CMI001 to the IEEE. → *has submitted(aicc , cmi001)* *has submitted cmi001 to (aicc ,ieee)*
nsubj(X/JJ, Y/NN), prep_K(X/JJ, A/NN)→ X_K(Y,A) (14)	The RTE describes the LMS requirements for managing the runtime environment such as standardized data model elements used for passing information relevant to the learner's experience with the content). → *relevant_to(information, experience)*
Nsubj(X/JJ, Y/NN), cop(X/JJ, V/VB) prep_P(X/JJ, Z/NN)→X_P(Y,Z)(15)	These branches are visible to the LMS. → *visible to(branch, lms)*
nsubj(X/JJ, Y/NN), cop(X/JJ, C/VB), xcomp(X/JJ, V/VB), dobj(V/VB, Z/NN), → X_V(Y, Z) (16)	The Sequencing Control Choice element indicates that the learner is free to choose any activity in a cluster in any order without restriction. → *free to choose (learner, activity)*
nsubj(X/JJ, Y/NN) xcomp(X/JJ, V/VB) agent(V/VB, Z/NN)→ X_V(Y,Z) (17)	The difficulty lies in the fact that the set of all possible behaviors given all possible inputs is too large to be covered by the set of observed examples. → *too large to be covered by (set of possible behavior, set of observed examples)*
Nsubj(X/VB, Y/NN), dobj(X/VB, V/NN) xcomp(X/VB, Z/VB), dobj(Z/VB, N/NN)→ *X_V_Z(Y, N)*(18)	SCORM recognizes that some training resources may contain internal logic to accomplish a particular learning task → *may contain internal logic to accomplish (training resource, learning task)*

Passive clauses.

Passive clauses allow the extraction of conceptual relationships (Table 4) and sometimes their inverse property. For instance, pattern (19) (Table 4) can be used to define such an inverse property for the relation *defined set of information - can be tracked by - lms environment* by creating an OWL inverseOf relation: *lms environment - can track - defined set of information.*

Table 4. Passive clauses patterns for conceptual relationship extraction

Pattern	Example
`Nsubjpass(V/VB, Y/N), Agent(V/VB, Z/N)→ V(Y,Z) (19)`	The purpose of establishing a common data model is to ensure that a set of information can be tracked by different LMS environments. → *can be tracked by (defined set of information, lms environment)*
`Nsubjpass(V/JJ, X/NN), cop(V/JJ, V/VB), prep_P(V/JJ, Y/NN)→ V_P(X,Y)(20)`	Sequencing information is defined on the Activities and is external to the training resources associated with those Activities. → *external to (sequencing information, training resource)*
`Nsubjpass(V/VB, X/NN), prep_P(V/VB, Y/NN)→ V_P(X,Y) (21)`	Metadata can be applied to Assets. → *can be applied to (metadata, assets)*
`Nsubjpass(V/VB,X/NN), dobj(V/VB,Y/NN)→ V(X,Y)(22)`	The data model element names shall be considered reserved tokens. → *shall be considered(data model element names, reserved tokens)*

Relative Clauses.

Relative clauses (see Table 5) are generally neglected by similar pattern-based approaches, as they are often distant from their main subject. The relationships created by our patterns in this category have generally lengthy labels, but they allow us to find links between two candidate concepts that might be otherwise neglected.

Table 5. Relative clauses patterns for conceptual relationship extraction

Pattern	Example
`rcmod(X/NN, Y/VB), dobj(Y/VB, Z/NN), prep_P(Y/VB, Q/NN) → Y_Z_P(X, Q), Y(X, Z) (23)`	Learning Management System is a software that automates event administration through a set of services. → *automates_event_administration_through (Software, set_of_services)*
`Nsubj(X/NN, Y/NN), rcmod(X/NN, V/VB), dobj(V/VB, Z/NN)→V(Z,Y)(24)`	→ *automates(learning management system, event administration)*
`Nsubj(X/NN, Y/NN), rcmod(X/NN, V/VB), dobj(V/VB, Z/NN), Prep_P(V/VB, Q/NN)→V_Z_P(Y, Q)(25)`	→ *automates event administration through (Learning management system, set of services)*

`Rcmod(X/NN, V/VB), xcomp(V1/VB,V2/VB), dobj(V2/VB, Y/NN), prep_P(V2/VB, Z/NN)→ V1_V2_Y_P (X, Z) (26)`	"1484.11.1" is a standard that defines a set of data model elements that can be used to communicate information from a content object to an LMS. → *can be used to communicate information from (set of data model element, content object)*
`rcmod(X/NN, V/VB), dobj(V/VB, Z/NN)→ V(X,Z) (27)`	This keyword data model element can only be applied to a data model element that has children. → *has (data model element, children)*

Relative clauses patterns are focused around the dependency relationship *rcmod* and enable making links between a main subject and the *rcmod* direct object (Pattern 24) or a preposition (Pattern (23) and (25)) in the relative clause. Pattern (26) makes a link between the noun in a relative clause modifier with the clausal complement xcomp, and finally Pattern (27) links the noun of the relative clause modifier to its direct object.

Other clauses.

Finally, there are some clauses around infinitival modifiers (*infmod*) and participial modifiers (*partmod*) that modify their noun phrase and that allow us to create conceptual relationships (Table 6) when they have a direct object or a preposition. Note again that pattern 31 (Table 6), which has an agent dependency, can lead to an OWL inverse property similar to the one explained in the passive clauses.

Table 6. Other clauses patterns for conceptual relationship extraction

`Infmod(X/NN,Y/VB), Dobj(Y/VB,Z/NN)→Y(X,Z) (28)`	This value can be requested by the SCO to determine the next index position.→ *determine (sco, next index position)*
`Partmod(X/NN,V/VB),dobj(V/VB, Y/NN)→ V(X,Y) (29)`	A SCO can communicate with an LMS using the SCORM RTE → *using(lms, scorm rte)*
`Partmod(X/NN, V/VB), prep_P(V/VB, Y/NN) → V_P(X,Y) (30)`	...to describe the components used in a learning experience. → *used in (components, learning experience)*
`Partmod(X/NN, V/VB), agent(V/VB, Y/NN)→V(X,Y) (31)`	All data model elements described by SCORM are …. *described by (data model elements, scorm)*

4 Evaluation

In order to evaluate our patterns, we essentially relied on two corpora used in previous experiments [6]: the SCORM corpus, which is a set of manuals on the SCORM eLearning standard and the AI corpus, which is a set of Wikipedia pages about artificial intelligence. We previously generated and validated two OWL ontologies from these corpora and we consider them as our gold standards (GSs). Details about these GSs can be found in [6][4]. We then calculated the precision of the various patterns based on these GSs.

Precision= number of generated relationships or concepts per pattern (A) / number of relationships or concepts in (A) that exist in the GS

Tables 7-11 report the various results of each pattern category[5]. Each table presents, for each pattern in the category, the precision of the extracted relationships and concepts in both corpora. One must note that some of the relationships extracted by patterns were perfectly valid (from a lexical point of view) but were not found in the GS, thus reducing the reported precision. Another point is that concepts are simple or multi-words expressions that occur in a relationship. Therefore, we were able to calculate concept precision as well by identifying how many concepts involved in the extracted relationships were in the GSs.

Table 7. Precision of Hierarchical Relationships Patterns.

Pattern	Relations SCORM	Relations AI	Concepts SCORM	Concepts AI
Pattern (7)	47.46	93.75	79.45	84.37
Pattern (11)	74.63	56.76	91.29	72.79
Pattern (8)	100.00	76.92	100.00	80.77
Average	**74.03**	**75.81**	**90.25**	**79.31**

We can notice that the precision of hierarchical relationships and their corresponding concepts is quite high (Table 7).

Table 8. Precision of Conceptual Relationships Patterns (Main Clauses)

Pattern	Relations SCORM	Relations AI	Concepts SCORM	Concepts AI
Pattern (12)	52.16	24.10	81.75	57.45
Pattern (16)	75.00	0	100.00	33.33
Pattern (15)	50.75	35.29	79.58	61.67
Pattern (13)	50.00	26.25	78.24	49.51
Pattern (14)	36.31	28.33	79.49	62.62

[4] Available at http://azouaq.athabascau.ca/goldstandards.htm

[5] Extraction examples for each pattern can be found at http://azouaq.athabascau.ca/experiments/wop2012/SCORMPatterns_WOP2012.xls and AI-Patterns_WOP2012.xls

Pattern (17)	0	0	0	0
Pattern (18)	0	0	80.43	90
Average	**37.74**	**16.28**	**71.35**	**50.65**

Interestingly, passive clauses (Table 9) and other clauses (Table 11) achieve a better performance for relationships precision than main clauses (Table 8) which get approximately the same results as relative clauses (Table 10).

Table 9. Precision of Conceptual Relationships Patterns (Passive Clauses).

Pattern	Relations SCORM	Relations AI	Concepts SCORM	Concepts AI
Pattern (19)	69.56	25.00	74.00	56.25
Pattern (22)	60.00	66.67	88.00	100.00
Pattern (21)	58.45	44.78	84.16	66.67
Pattern (20)	75.00	*na*	95.00	*Na*
Average	**65.75**	**45.48**	**85.29**	**74.31**

Table 10. Precision of Conceptual Relationships Patterns (Relative Clauses).

Pattern	Relations SCORM	Relations AI	Concepts SCORM	Concepts AI
Pattern (23)	41.18	56.25	62.07	66.67
Pattern (25)	0	0	90.48	60.00
Pattern (24)	73.33	50.00	93.90	76.47
Pattern (26)	0	0	85.71	0
Pattern (27)	50.62	21.62	86.39	56.52
Average	**33.02**	**25.57**	**83.71**	**51.93**

Table 11. Precision of Conceptual Relationships Patterns (Other Clauses).

Pattern	Relations SCORM	Relations AI	Concepts SCORM	Concepts AI
Pattern (29)	43.18	25.00	80.58	70.00
Pattern (30)	60.58	54.54	86.04	73.58
Pattern (28)	38.77	37.50	82.10	80.77
Average	**47.51**	**26.85**	**82.91**	**74.78**

These results give a general idea on the Precision of the lexico-syntactic patterns. As we have previously mentioned, and as the results confirm it, there is a need to filter the various extractions using statistical and/or graph-based metrics [6]. The most frequent patterns were *nsubj-dobj*, *nsubjpas-prep*, *nsubj-dobj-prep*, *nsubj-prep* and *partmod-prep*. The most precise (but scarcer) patterns, without any filtering, were hierarchical patterns. One important observation is the quite high precision of concepts even without filtering. Regarding relationships, it is possible to imagine that if concepts of interest are known upfront, then these patterns will be very useful for discovering relationships between these predefined concepts. This will be tackled in

future work. We also created few patterns for attributes extraction involving possessives or nominal subject and copula with adjectives. However, the way to translate these attribute relationships into OWL-DL was not straightforward.

5 Conclusion

This paper presented a list of the main patterns used in OntoCmaps, our ontology learning tool. These patterns target specific syntactic structures in a dependency representation and are useful for the extraction of multi-word expressions and triples that can be later translated into OWL classes and properties. There were some simplifying assumptions made in OntoCmaps, mainly the removal of determiners and the lack of co-reference resolution that should be included in future work. In this current state, our patterns represent a good starting base that any researcher in text mining might use and especially the ontology learning community which lacks clear and reusable design patterns. Overall, future efforts will tackle how and if a more fine-grained semantic analysis would be beneficial to the ontology learning task. Another future task will be to extend the coverage of our patterns by extracting frequently occurring syntactic structures using machine learning methods. Finally, one of the lessons learned in this paper is that such pattern-based extraction should necessarily be coupled with a filtering mechanism to increase the precision of the extractions.

Acknowledgments. This research was funded by the NSERC Discovery Grant Program.

References

1. Blomqvist. E.: Semi-automatic Ontology Construction based on Patterns. PhD Thesis. Linköping University, Department of Computer and Information Science (2009)
2. Presutti, V. et al.: A Library of Ontology Design Patterns: Reusable Solutions for Collaborative Design of Networked Ontologies. NeOn D2.5.1 (2008)
3. Völker, J., Hitzler, P. & Cimiano, P.: Acquisition of OWL DL axioms from lexical resources. In: Proc. of the 4th European Semantic Web Conf., pp.670-685, Springer (2007)
4. Hearst. M.: Automatic Acquisition of Hyponyms from Large Text Corpora. In Proc. of the 14th International Conf. on Computational Linguistics, pp.539-545 (1992).
5. De Marneffe, M-C , MacCartney B. & Manning, C. D.: Generating Typed Dependency Parses from Phrase Structure Parses. In Proc. of *LREC,* pp. 449–454 (2006).
6. Zouaq, A., Gasevic, D. & Hatala, M.: Towards open ontology learning and filtering. Inf. Syst. 36(7): 1064-1081 (2011)
7. Maynard, D.; Funk, A. & Peters, W.: Using Lexico-Syntactic Ontology Design Patterns for ontology creation and population, CEUR-WS.org, (2009).
8. Zouaq, A., Gagnon, M. & Ozell, B.: Semantic Analysis using Dependency-based Grammars and Upper-Level Ontologies, *International Journal of Computational Linguistics and Applications*, 1(1-2): 85-101, Bahri Publications (2010).
9. Toutanova, K., Klein, D., Manning, D. & Singer. M.: Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proc. of HLT-NAACL*, pp. 252-259 (2003).
10. Snow, R., Jurafsky, D. and Ng, A. Y.: *Learning syntactic patterns for automatic hypernym discovery.* In: Advances in Neural Information Processing Systems, pp.1297-1304 (2004).

Applications of Ontology Design Patterns in the Transformation of Multimedia Repositories

Agnieszka Ławrynowicz[1] and Raúl Palma[2]

[2]Poznan University of Technology, Poznań, Poland alawrynowicz@cs.put.poznan.pl
[2]Poznan Supercomputing and Networking Center, Poznań, Poland
rpalma@man.poznan.pl

Abstract. The development of ontologies from scratch is a very inefficient approach. Hence, ontology development is increasingly being conducted by reusing existing ontological and non-ontological resources as it lowers the time and cost of developing new ontologies, avoids duplicating efforts and ensures interoperability. Similarly, the emergence of ontology design patterns has facilitated the reuse of best practices in ontology engineering, improving the quality of the developed ontology. In this paper we show how different ontology design patterns along with state-of-the-art ontology engineering methodologies help during the construction of ontologies, by describing the development process of the Digital Multimedia Repositories Ontology (DMRO). We show the applicability of the developed ontology by using it as the basis for the transformation of Videolectures multimedia repository to RDF data.

1 Introduction

The widespread use of ontologies in different applications and domains in the last years has led to an increasing interest of researchers and practitioners in the development of ontologies. To speed up the development process, existing resources in the form of ontologies and other non-ontological forms, such as thesauri, lexica and DBs, as well as best practices encoded in the form of ontology design patterns (ODPs) are increasingly being used. Reusing existing resources is an important aspect that is progressively better supported by the growing availability of ontology design patterns and ontologies for different domains, the availability of well established upper-level ontologies, and better support for transformation of non-ontological resources to ontological format. Besides speeding up the development process, reusing existing resources has many benefits, including lower development costs, interoperability with other ontologies and better quality of the developed ontology. Even more, we argue that in addition to the availability of such resources, ontology engineers can benefit from the availability of good use cases showing how to apply all of these resources in practical applications.

For example, one of the pilot applications of e-LICO project[1] aimed at describing multimedia artifacts in digital multimedia repositories, in terms of their usage, reviews and content type, as well as their relation with related events and agents, that would serve as background knowledge to be used in semantic data mining tasks. Although there are different ontologies available for describing multimedia artifacts (e.g.,

[1] http://www.e-lico.eu/

COMM, M3O), they focus mainly on a description of the artifacts characteristics with a detail that falls outside the application requirements, instead of focussing on the usage information of these resources which is key for data mining in multimedia repositories. Additionally, we can find ontologies and vocabularies describing parts of the required knowledge, such as agents or events (e.g., FOAF, SWRC), but they are generic or conceived for other applications, and are not related to multimedia resources. Consequently we developed the Digital Multimedia Repositories Ontology (DMRO), but reusing as much as possible knowledge from these and other ontologies and vocabularies.

Therefore, the goal of this paper is twofold: (i) to present the Digital Multimedia Repositories Ontology and show how it has been used in practice and (ii) to provide an example of the ontology development process where existing resources and ODPs were widely re-used. We show the applicability of the ontology by using it as the basis for the transformation of a multimedia repository coming from VideoLectures.Net portal to RDF data. VideoLectures.Net[2] is one of the largest scientific and educational video Web sites, mostly hosting lectures given by scholars and scientists at conferences, summer schools, and other events. The dataset used in our application of DMRO was prepared for the e-LICO data mining challenge [11] on recommender systems for the lectures.

The rest of this paper is structured as follows: Section 2 discusses the development of DMRO, Section 3 describes the modules of DMRO, Section 4 describes the application of DMRO in transformation of Videolectures.net data, Section 5 discusses the related work, Section 6 contains lessons learnt, and Section 7 concludes the paper.

2 Development of DMRO

After an analysis of existing ontology engineering methodologies (see [8,2] for a detailed comparison), we decided to follow the NeON Methodology for Building Ontology Networks ([7,2]) during the development of DMRO for several reasons: (i) its flexibility and adaptability to different development scenarios, specially those with focus on reusing (and reengineering) existing knowledge resources as well as best practices in ontology engineering ; (ii) the clear guidelines provided for every task with concise information cards, templates, heuristics and examples; (iii) and the technological support available for it through the NeOn Toolkit[3].

2.1 Requirements

In line with this methodology, we started the development of DMRO by collecting and analyzing an initial set of requirements using a structured document, called the Ontology Requirements Specification Document (ORSD)[4]. The ORSD document covers the following topics concerning the ontology: purpose, scope, implementation language, intended end-users, intended use cases, reusing ontology statements.

In particular, the goal of the engineering of DMRO is to use it for the tasks of: a) design of recommendation and personalization solutions for digital multimedia repositories; b) meta-learning/meta-mining on data mining experiments repositories; c) testing

[2] http://videolectures.net/
[3] http://neon-toolkit.org/
[4] Available at http://129.194.69.119/public/dmro/DMRO_ORSD.pdf

semantic data mining [9] algorithms. The scope of the ontology is on the applications in recommender systems, personalization, and adaptive faceted browsers of digital resources, and therefore the ontology being built is an application domain ontology.

DMRO should be as compatible as possible with established ontologies and vocabularies that cover relevant aspects in the domain of DMRO, such as Dublin Core[5], FOAF[6], RDF Review[7], OBO Relation Ontology[8], and OAI-ORE[9]. Consequently, it should reuse terms from these resources whenever is possible.

Additionally, in order to encourage the re-use of DMRO and facilitate its specialization for particular applications, it should follow a modularized approach, thus it is expected to build a set of small modules for representing DMRO concepts, such as: multimedia resources, users, events, reviews, Web Usage Mining related concepts, and the domain topics (in our case VideoLectures.Net topic category). Similarly, in order to ensure compatibility with existing tools and vocabularies, the current standard OWL 2 should be used as the implementation language.

The development was scheduled in two cycles/iterations within the lifetime of the e-LICO project. Each of these cycles was scheduled using the Gontt tool[10], which enables the graphical representation of an ontology project schedule in the form of a Gantt chart. Figure 1 illustrates the schedule for the first cycle.

The next section describes the resources re-used during the ontology construction.

2.2 Reusing Existing Resources

From the analysis of ontologies relevant for our domain and application, we couldn't find one single ontology covering all aspects of DMRO, but we found several ontologies modeling parts that could be reused. Hence, we decided to reuse individual statements instead of a whole ontology. Moreover, we decided to use an upper ontology that will model generic concepts that can be specialized for DMRO. The advantage of using an upper ontology will reflect in better interoperability and foster the reusability of DMRO. We also re-used and applied different ontology design patterns relevant for DMRO, which allowed us to follow the best practices in ontology engineering when modeling DMRO. Finally, we re-used a non-ontological resource: a dataset based on the data from Videolectures.net portal.

Upper Ontology

We used DOLCE Ultralite (DUL)[11] as an upper ontology. DUL is a very light version of DOLCE and DnS, which provides a simplification and an improvement of some parts of DOLCE Lite-Plus library[12], and Descriptions and Situations ontology [13]. DUL provides a set of upper level concepts that can be the basis for easier interoperability

[5] http://dublincore.org/documents/dcmi-terms/
[6] http://xmlns.com/foaf/spec/
[7] http://vocab.org/review/terms.html
[8] http://obofoundry.org/ro/
[9] http://www.openarchives.org/ore/
[10] http://neon-toolkit.org/wiki/Gontt
[11] http://ontologydesignpatterns.org/wiki/Ontology:DOLCE%2BDnS_Ultralite
[12] http://dolce.semanticweb.org
[13] http://www.ontologydesignpatterns.org/wiki/Ontology:DnS

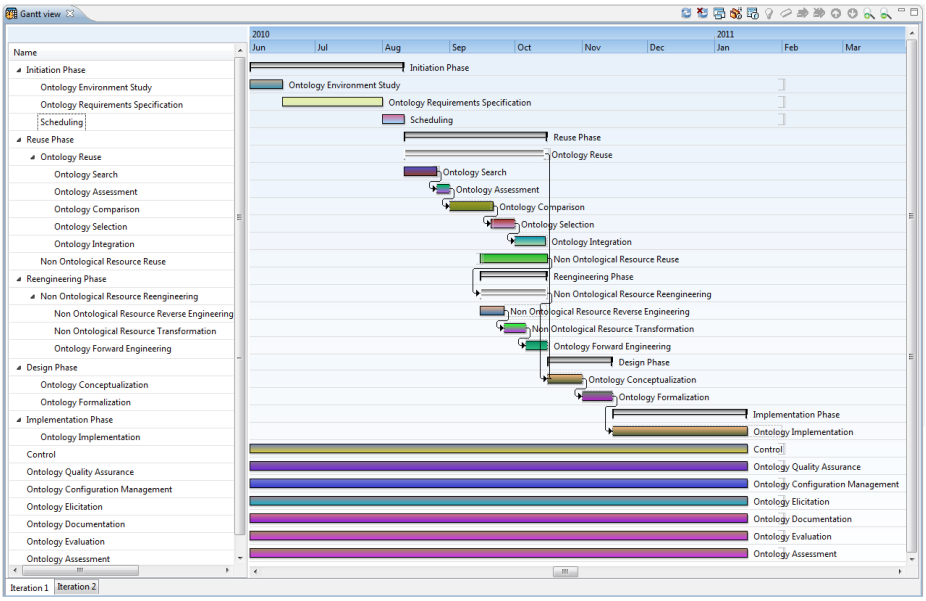


Fig. 1. First cycle of the development process of DMRO

among many middle and lower level ontologies. DOLCE-Ultralite falls within OWL-DL language, but it only uses OWL-Lite and disjointness constraints.

Ontology Statements

We identified relevant statements to be re-used from the following ontologies and vocabularies: (i) Dublin Core for modeling general metadata properties of resources; (ii) FOAF - for modeling the users, authors, participants, their personal data, and social aspects; (iii) SWRC ontology - for events; (iv) OBO Relation Ontology - for standard relations; (v) DCTYPE - for modeling collections; (vi) ORE - for modeling aggregations of resources; and (vii) RDF Review Vocabulary - for the reviews, comments and feedback. Additionally, we identified the following modules from the myExperiment Ontology: (viii) Viewings & Downloads - for modeling the usage of resources; (ix) Annotations - for modeling tags.

Finally, we have used IOLite extension of DUL[14] and CSnCs ontology[15], as an inspiration for modeling DigitalResources. IOLite is an ontology of information objects and realizations, plugin to DOLCE-Ultralite. CSnCS (Computer Science for Non-Computer Scientists) is a knowledge hierarchical repository of concepts in the domain of Information Technology for End Users, which also uses DUL as an upper ontology.

In order to select the statements to be reused we conducted a research to identify candidate ontologies, partially with the help of Watson plugin for NeOn Toolkit[16] (Figure 2 shows statements from CSnCS regarding DigitalResources), and followed a

[14] http://www.loa-cnr.it/ontologies/IOLite.owl
[15] http://www.let.uu.nl/lt4el/content/files/CSnCSv0.01Lex.zip
[16] http://neon-toolkit.org/wiki/Watson_for_Knowledge_Reuse

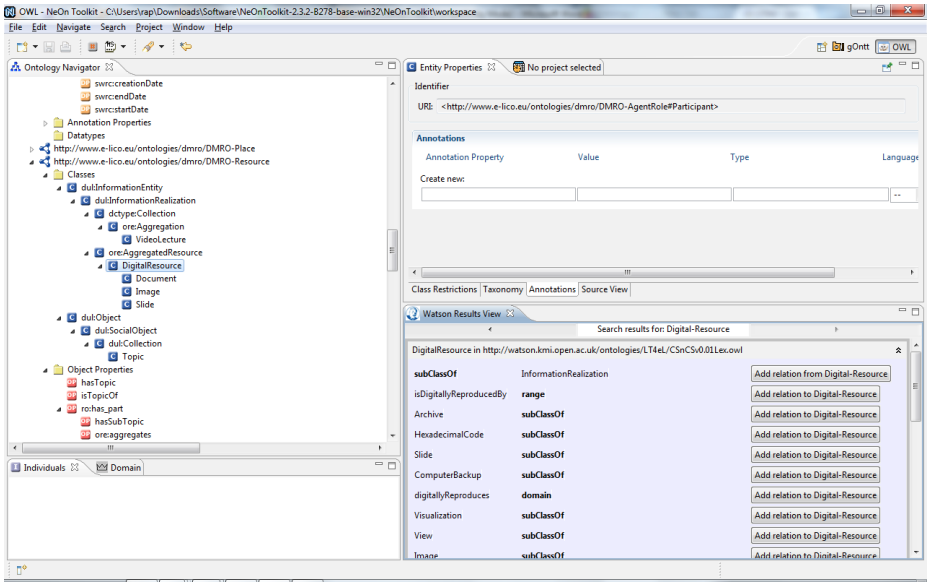


Fig. 2. Ontology statements search result in NeOn Toolkit

selection criteria consisting on the simplicity, coverage and popularity of the candidate ontologies, building from the dimensions proposed by the NeOn methodology for selecting ontological resources (i.e., understandability, integration effort and reliability).

Ontology Design Patterns

We re-used the following ontology design patterns from http://ontologydesignpatterns.org:

- Agent role[17] to represent agents and the roles they play. It was used with the following competency questions:
  - which agent does play this role?
  - what is the role that played by that agent?
- Participant role[18] to represent participants in events holding specific roles in that particular event. It was used with the following competency questions:
  - What is the role of this object in this event?
  - What is the object holding this role in this event?
  - In what event did this object hold this role?
- Tagging[19] to represent a tagging situation, in which someone uses a term, from a list of a folksonomy, to tag something (or the content of something). It was used with the following competency questions:
  - Who is tagging (the content of) what?
  - By using what term from what folksonomy?

[17] http://ontologydesignpatterns.org/wiki/Submissions:AgentRole
[18] http://ontologydesignpatterns.org/wiki/Submissions:ParticipantRole
[19] http://ontologydesignpatterns.org/wiki/Submissions:Tagging

  - Which polarity has the tagging?
- Place[20] to talk about places of things. It was used with the following competency questions:
  - Where is a certain thing located?
  - What is located at this place?
- Topic[21] to represent topics and their relations. It was used with the following competency questions:
  - What is the topic of something?
  - What topics are included in this one?
  - What are the topics near to that one?

Non-ontological resources

We re-used a dataset based on the data snapshot from VideoLectures.net taken in August 2010, which consisted of a database dump into several CSV files, which contained data from 7 different tables (authors, categories, events, lectures_train, lectures_test, authors_lectures and categories_lectures). The snapshot contained 8,105 video lectures, where 5,286 lectures were manually categorized into a taxonomy of around 350 scientific topics. This dataset was used as background knowledge during the specification phase of DMRO development. In particular, the competency questions and the pre-glossary of terms in the ORSD document were prepared largely on the basis of the dataset as an inspiration.

3 Overview of DMRO

DMRO consists of a set of 6 interrelated modules, which are imported from the main ontology file available at http://129.194.69.119/public/dmro/DMRO.owl. Next, we shortly describe each of the modules.

Resource. This module describes multimedia resources (see Fig. 3). Despite using DOLCE-Ultralite (DUL) as upper ontology, we have also used 'IOLite', extension of DUL, as an inspiration for modeling the concept *DigitalResources*. We also considered other ontologies such as COMM or M3O for modeling multimedia resources; however their focus on a detailed description of these resources falls outside the scope of our ontology as the ORSD shows. We have re-used 'Topic' ontology design pattern for modeling the topics, and statements from DCTYPE, ORE and RO. We have also used ontology CSnCs 'Computer Science for non-Computer Scientists from Project LT4eL (http://www.lt4el.eu/)' as inspiration.

Event. This module describes events, and was created using DUL as upper ontology, and by re-using statements from SWRC and DC (see Fig. 4). Additionally, we have specialized the *SWRC:Lecture* class with various types of lectures identified during the specification phase.

Agent-Participant-Role. The module describes agents and their roles as participants in events, based on the ParticipantRole ontology design pattern (see Fig. 5). Besides using DUL as upper ontology, we have re-used statements from FOAF and

[20] http://ontologydesignpatterns.org/wiki/Submissions:Place
[21] http://ontologydesignpatterns.org/wiki/Submissions:Topic

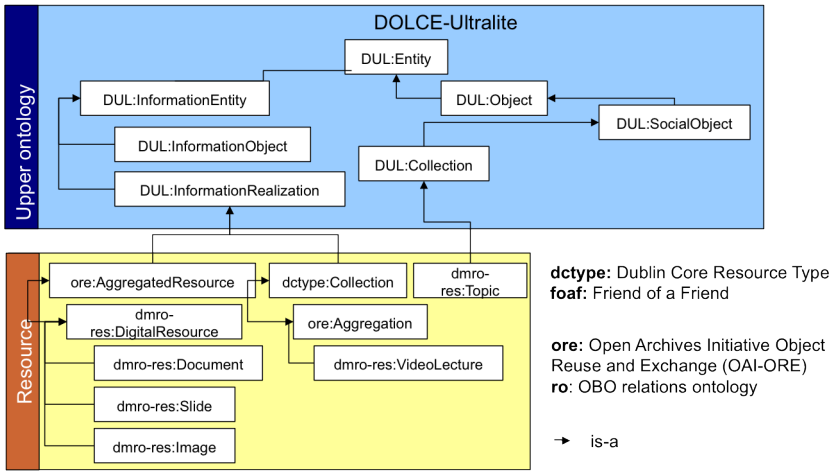


Fig. 3. An illustration of the part of DMRO's Resource module.

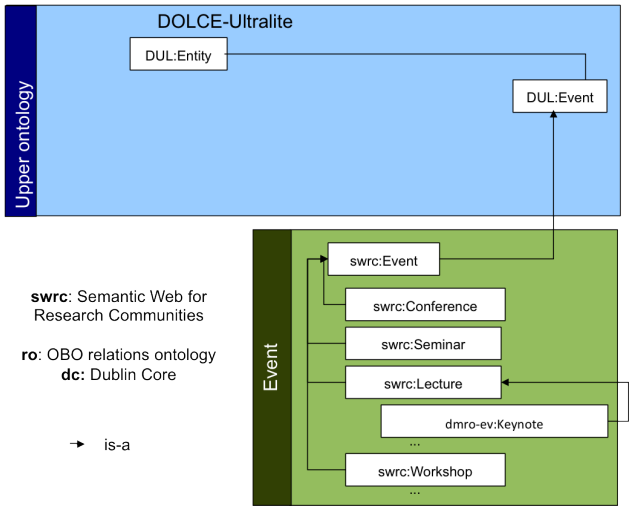


Fig. 4. An illustration of the part of DMRO's Event module.

RO. For example, instead of using the class *DUL:Agent* (which is a subclass of *DUL:Object*), we reused the more popular *FOAF:Agent* class and its subclasses. Referring to the suggestions of N.F. Noy and D.L. McGuinness on "an instance or a class" discussion[22] we decided to model agent roles, such as author and participant (e.g. dmro-apr:Presenter, dmro-apr:Author) as instances and not classes in the ontology as these are the most specific elements that are going to be represented in the knowledge base, i.e., they constitute the most specific elements in the answer to the competency

[22] http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

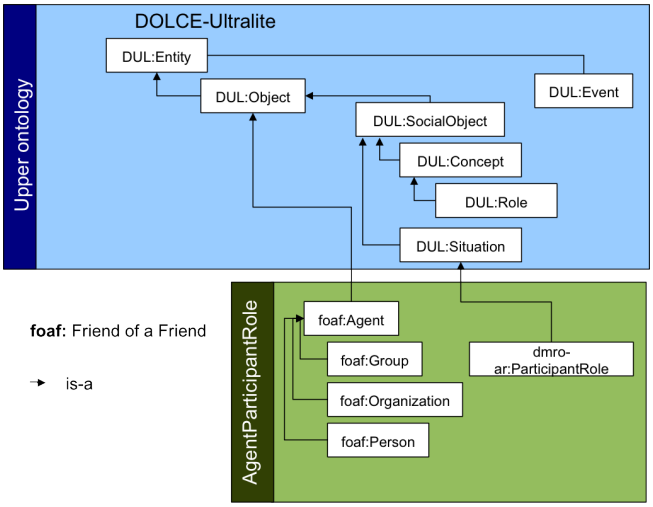


Fig. 5. An illustration of the part of DMRO's Agent-Participant-Role module.

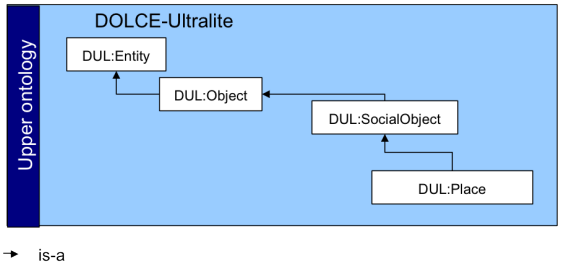


Fig. 6. An illustration of the part of DMRO's Place module.

questions related to agents roles (e.g., what is the role that played by that agent?). Before being taken, this design decision was discussed with the help of Cicero, an argumentation tool that is part of the e-LICO collaborative ontology development platform.

Place. This ontology module represents generic locations and was created based on the Place ontology design pattern, which is already implemented in DUL (see Fig. 6). So, we reused the relevant DUL statements and also some statements from RO (e.g., *located_in* property)

Review. This module was created by re-using the RDF Review Vocabulary and using DUL as upper ontology (see Fig. 7). We have also re-used statements from FOAF, DC and myExperiment Ontology. Mainly, the classes *Comment*, *Feedback* and *Review* from the RDF Review Vocabulary have been modeled as types of annotations (as in myExperiment Ontology).

Annotation. This ontology module was created by reusing statements from the Viewings & Downloads and Annotations modules of the myExperiment Ontology to model different resource annotations, including usage information and tagging, and us-

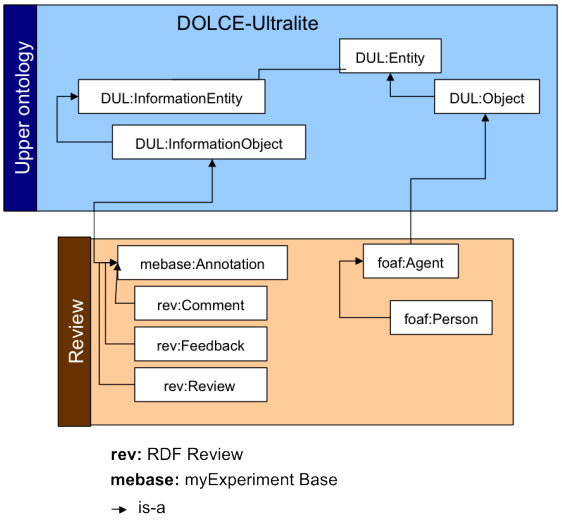


Fig. 7. An illustration of the part of DMRO's Review module.

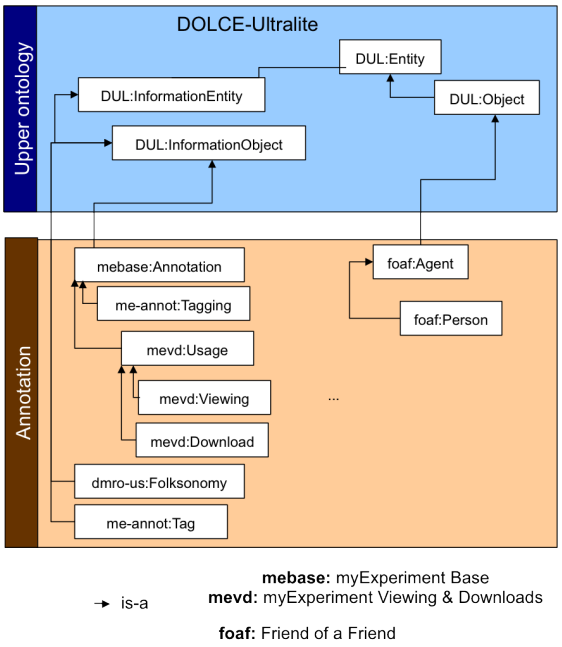


Fig. 8. An illustration of the part of DMRO's Annotation module.

ing DUL as an upper ontology (see Fig. 8). We also reused statements from the RO ontology, and some concepts from the Tagging ontology design pattern.

Table 1. Mapping for author table of Videolectures.net

Column	Ontology Element	Element Type	OWL Axiom Type
id	foaf:Person	Individual	Declare & ClassAssertion
name	foaf:name	DataProperty	DataPropertyAssertion
email	DMRO-AgentParticipantRole:email	DataProperty	DataPropertyAssertion
organization	foaf:Organization	Individual & ObjectProperty & DataProperty	ClassAssertion & ObjectPropertyAssertion & DataPropertyAssertion

4 Application of DMRO

DMRO has been used as the basis for (i) representing Videolectures.net topic hierarchy and (ii) transforming Videolectures.net dataset into RDF. The former was a simple test during which we generated instances of the concept *Topic* and re-used properties from SKOS vocabulary for the representation of hierarchical links (i.e., *skos:broader* and *skos:narrower*). The resulting knowledge base is available at http://129.194.69.119/public/dmro/DMRO-VLNetCat.owl.

For the transformation of Videolectures.net dataset into RDF we generated manually mappings between the terms from DMRO and terms from the dataset (e.g., columns in tables). The mappings range from simple alignments between a table column and a dataProperty in the ontology, to more complex alignments that created several ontology axioms for a value in a table column. Table 1 contains sample mapping that was created for the authors table[23]:

During preparation of the mapping, we have introduced several changes into DMRO (some of them are discussed in the Cicero argumentation tool within e-LICO collaborative ontology development platform) which allowed us to represent more accurately Videolectures.net data, and more generally digital multimedia repositories. The actual transformation was performed programmatically based on the conceptual mapping specified. The resulting knowledge base is available at http://129.194.69.119/public/dmro/DMROKB.owl and has already been used to test semantic data mining methods.

5 Related Work

Previous efforts in the multimedia domain, have led to the development of different ontologies, although most of them have been mainly focused on the detailed description of multimedia artifacts and only in few cases ontology design patterns were reused during the development process. For instance, COMM ontology [1], which is composed of multimedia patterns covering different media types (e.g., visual, audio or text), is based on DOLCE foundational ontology and the MPEG-7[24] standard. COMM reused two main design patterns: Descriptions & Situations (D&S) [5] and Ontology of Information Objects (OIO) [4], which were specialized to create multimedia design patterns

[23] The complete set of mappings is available at http://129.194.69.119/public/dmro/mapping.xls
[24] http://mpeg.chiariglione.org/standards/mpeg-7/mpeg-7.htm

for decomposition of multimedia content into segments, the annotation of these segments, as well as basic patterns to formalize the notion of digital data and algorithm. M3O [3] aimed for rich presentations in the web (SMIL, SVG, Flash), covers multimedia, audio/music, image, video, and re-uses DUL. M3O also followed a pattern-based approach to ontology design, and used six patterns: Decomposition, Annotation, Information Realization, Data Value for representing complex values, Collection, and Provenance. Although these ontologies were useful examples in the domain, they were found too detailed and focused in describing multimedia resources, which was not the goal of DMRO as discussed in the introduction.

Other relevant ontologies include the Media Resource Ontology [12], a W3C initiative aimed at integrating data resources related to media on the Web that covers multimedia, audio/music, video, and provides mappings to various multimedia metadata formats (e.g., MPEG-7). Similarly, we can find also various initiatives that focus on the transformation of MPEG-7 standard to ontological format, such as [10] and [6].

6 Discussion

The NeON methodology was very useful in identifying relevant existing resources. The competency questions facilitated identification of relevant ODPs as well as ontologies to re-use. In the latter case, we found that pre-glossary of terms (a part of an ORSD document) is especially useful for this task. The terms extracted from competency questions (and answers to them) allowed us to efficiently search for ontologies.

The major lesson learnt and an idea for improvements concerns further re-using of identified ODPs and ontologies and deals with availability of guidelines on a vocabulary that could be used to instantiate or specialize chosen ODPs. Since ODP entities in principle constitute templates, they need to be further refactored while being incorporated to a developed ontology. This needs modeling decisions concerning the choice of a vocabulary and namespaces. We would find it very useful if such guidelines existed, for instance, pointing to most popular terms that are used by ontology engineers to instantiate a given ODP, possibly indicating also the domain of the ontology where they were used. The information on the domain is important due to differences in term popularity across different domains w.r.t. the same pattern. This would allow not only to re-use an ODP, but also to: i) refactor it such that most widespread vocabulary is used or/and ii) proper vocabulary for given domain is used. Similarly, we think that re-using ODPs would be greatly facilitated if example ontologies re-using a given ODP where listed, most preferably together with the information on a domain.

Currently, http://ontologydesignpatterns.org has placeholders for 'Known uses' of an ODP and for 'Examples (OWL files)', but they are often empty. Besides of that, we would like to stress the aspect of popularity of "known uses" and "examples" in the context of a given domain to make it easier for an ontology engineer to choose proper vocabulary. For instance, we chose to use myExperiment for representing tagging since we found the annotations ontology of myExperiment covering many concepts we wanted to model in the Annotation module, including tags. However, as one of the reviewers suggested, we could have rather used the Tag ontology (http://www.holygoat.co.uk/owl/redwood/0.1/tags/) which is already widespread in Linked

Data on multimedia such as in DBTune datasets, if we were aware of its popularity. Listing it as an example accompanying the ODP would help in making this choice.

7 Conclusions

This paper presented the Digital Multimedia Repositories Ontology and its application in the transformation of the data from Videolectures.net into a knowledge base represented in RDF. Importantly, by discussing the ontology development process and its application, it also provided a use case of re-using existing ontological and non-ontological resources and ODPs for ontology construction.

The DMRO-based RDF version of the Videolectures.net dataset provides proof-of-concept of the coverage of the DMRO terms, and their suitability to represent required knowledge on the Videolectures.net use case. It may be used to test semantic data mining approaches, some of them already developed within e-LICO, as well as in the experiments in the digital multimedia repositories.

Acknowledgments The research reported in this paper has been supported by the EU e-Lico project (http://www.e-lico.eu/, FP7 STREP ICT-231519).

References

1. R. Arndt, R. Troncy, S. Staab, L. Hardman, and M. Vacura. COMM: Designing a well-founded multimedia ontology for the Web. In *In Proceedings of the 6th International Semantic Web Conference (ISWC2007), Busan, Korea*, pages 11–15, 2007.
2. M. del Carmen Suárez-Figueroa. *NeOn Methodology for Building Ontology Networks: Specification, Scheduling and Reuse*. PhD thesis, Universidad Politécnica de Madrid, Spain, 2010.
3. D. Eißing, A. Scherp, and C. Saathoff. Integration of existing multimedia metadata formats and metadata standards in the M3O. In *Proceedings of the 5th international conference on Semantic and digital media technologies*, SAMT'10, pages 48–63, Berlin, Heidelberg, 2011. Springer-Verlag.
4. A. Gangemi, S. Borgo, C. Catenacci, and J. Lehmann. Task taxonomies for knowledge content D07. Technical report, Metokis Project, 2004.
5. A. Gangemi and P. Mika. Understanding the Semantic Web through Descriptions and Situations. In *Proceedings of ODBASE03 Conference*, pages 689–706. Springer, 2003.
6. R. Garca and O. Celma. Semantic integration and retrieval of multimedia metadata. In S. Handschuh, T. Declerck, and M. Koivunen, editors, *Proceedings of the ISWC 2005 Workshop on Knowledge Markup and Semantic Annotation (Semannot'2005)*, volume 185, pages 69–80. CEUR Workshop Proceedings, 2005.
7. A. Gómez-Pérez and M. del Carmen Suárez-Figueroa. Scenarios for building ontology networks within the NeOn methodology. In *K-CAP*, pages 183–184, 2009.
8. A. Gomez-Perez, M. Fernandez-Loez, and O. Corcho. Methodologies and methods for building ontologies. In *Ontological Engineering*, Advanced Information and Knowledge Processing, pages 107–197. Springer Berlin Heidelberg, 2004.
9. M. Hilario, A. Kalousis, N. Lavrac, A. Lawrynowicz, J. Potoniec, and A. Vavpetic. Semantic data mining tutorial at ECML/PKDD'2011.
10. J. Hunter. Adding multimedia to the Semantic Web: Building an MPEG-7 ontology. In *SWWS*, pages 261–283, 2001.
11. T. Šmuc, N. Antonov-Fantulin, and M. Morzy, editors. *Proceedings of the ECML/PKDD Discovery Challenge Workshop 2011*, Athens, Greece, September 2011.
12. www.w3.org. Ontology for media resource 1.0. http://www.w3.org/TR/mediaont-10/, 2012.

SPARQL-DL Queries for Antipattern Detection

Catherine Roussey[1], Oscar Corcho[2], Ondřej Šváb-Zamazal[3], François Scharffe[4], and Stephan Bernard[1]

[1] Irstea, 24 Av. des Landais, BP 50085, 63172 Aubiére, France
catherine.roussey@irstea.fr
[2] Ontology Engineering Group, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid, Spain
ocorcho@fi.upm.es
[3] Knowledge Engineering Group, University of Economics Prague, Czech Republic
ondrej.zamazal@vse.cz
[4] LIRMM, Université de Montpellier, France
scharffe@lirmm.fr

Abstract. Ontology antipatterns are structures that reflect ontology modelling problems, they lead to inconsistencies, bad reasoning performance or bad formalisation of domain knowledge. Antipatterns normally appear in ontologies developed by those who are not experts in ontology engineering. Based on our experience in ontology design, we have created a catalogue of such antipatterns in the past, and in this paper we describe how we can use SPARQL-DL to detect them. We conduct some experiments to detect them in a large OWL ontology corpus obtained from the Watson ontology search portal. Our results show that each antipattern needs a specialised detection method.

Keywords: OWL, ontology, antipattern, SPARQL, SPARQL-DL

1 Introduction

In knowledge engineering, the concept of knowledge modelling pattern or ontology design pattern is used to refer to modelling solutions that allow solving recurrent knowledge modelling or ontology design problems, as defined in [14]. Antipatterns are defined as patterns that appear obvious but are ineffective or far from optimal in practice, representing worst practices about how to structure and design an ontology. However, in contrast to ontology design patterns, which are rooted deeply in the most recent ontology engineering methodologies, the work on antipatterns is less detailed. Antipatterns may have several applications: Antipatterns can be used to train and guide new ontology developers. Ontology editors can incorporate antipattern detection tool in order to detect potential errors during ontology development. Most of ontology systems that deal with a large set of ontologies, like ontology retrieval systems, need ontology quality measures. The quality of ontology can be evaluated by computing the number of antipattern occurrences. As far as we know antipattern studies are mainly applied to ontology debugging tasks. One of the earliest works in

this direction was set by the OntoClean method [10], which defined a set of meta-properties applied to classes and a set of procedures to check and correct the subsumption relations between classes. Other sources for antipatterns are: [19], which proposes four terminological patterns applied on class names to detect possible errors in subsumption relations between classes. The Laboratory of Applied Ontology has identified four logical antipatterns called MixedDomains, all of them focused on property domains and ranges. And [15] describes common difficulties for newcomers to Description Logics (DL) in understanding the logical meaning of expressions.

Several tools can be used for antipattern detection, most of which are available inside ontology editors and require the use of a reasoner to provide their justifications. Pellint[8] focuses on the detection and repair of antipatterns to improve ontology reasoning performance. The Protégé Explanation Workbench [11] and SWOOP [13] provide justifications of inconsistencies in ontologies based on the outputs of DL reasoners. However, using a reasoner for this purpose is not always possible, since in some complex ontologies, where the number of errors is too high, reasoners fail to provide any results. Besides, the catalogue of antipatterns or errors that they can detect is fixed.

Our antipattern detection methods follow a more general approach. They can work with an extensible set of antipatterns and some of them can be applied without the use of a reasoner. In general, our approach is based on the use of a set of SPARQL-DL queries for each antipattern to be detected. Then, each SPARQL-DL query is translated into SPARQL one. In our process, we can decide whether inferences are enabled or not before running any SPARQL queries, and we also offer the possibility of transforming the original ontologies into a form where SPARQL queries should retrieve more results.

We first tested our methods on the detection of one complex antipattern using only a set of SPARQL queries [16]. This first experiment was applied on 5 ontologies. This paper presents a larger experiments using a more generic approach: More antipatterns are detected on a larger set of ontologies. We also try to simplify the creation of queries using SPARQL-DL language. One of our final goal is to understand how often antipatterns appear in existing publicly available ontologies.

This paper is structured as follows. Section 2 briefly describes the antipatterns that will be used to run our experiments. Section 3 will describe the methods we have followed in order to run the experiments. Section 4 describes the experiment setup and the results of the experimentation. Finally, Section 5 provides some conclusions to the work done, based on the experiment results, and outlines the next steps to be done in our work.

2 A catalogue of antipatterns

A set of patterns commonly used by domain experts in their implementation of OWL ontologies are identified in [9]. These patterns resulted in unsatisfiable classes or modelling errors, due to misuse or misunderstanding of DL expressions.

In this section we will describe 4 antipatterns which are the ones that, as our experience has shown, are easier to understand and debug by domain experts. These patterns are categorized into two groups by [9]:

- *Detectable Logical AntiPatterns (DLAP)*: this type of patterns generates unsatisfiable classes that are normally identified by existing ontology debugging tools, although the information provided back to the user is not described according to such a pattern. This makes it sometimes difficult for ontology developers to find a good solution [11], [13].
- *Cognitive Logical AntiPatterns (CLAP)*: they represent possible modelling errors that may be due to a misunderstanding of the logical consequences of the used expression. This type of patterns is not detected by debugging tools, although in some cases their combination may lead into unsatisfiable classes that are detected.

Now we briefly[5] describe them from the simplest to the most complicated one. Each description contains:

- name, acronym and category that they belong to (i.e. DLAP or CLAP),
- several formal descriptions using the german syntax of DL [6],
- brief explanation of why this antipattern can appear.

SynonymOrEquivalence (SOE) antipattern – CLAP category

$$C_1 \equiv C_2; \tag{1}$$

The ontology developer wants to express that two classes C_1 and C_2 are identical, something which is not particulary useful especially if the ontology does not import others. Indeed, what the ontology developer generally wants to represent is a terminological synonymy relation, i.e. a class has two labels: the labels associated (or used as) the name of the classes C_1 and C_2. Usually one of these classes is not used anywhere else in the axioms defined in the ontology.

EquivalenceIsDifference (EID) antipattern – DLAP category

$$C_1 \equiv C_2;\ \ Disj(C_1, C_2); \tag{2}$$

$$C_1 \sqsubseteq C_2;\ \ Disj(C_1, C_2); \tag{3}$$

where the notation $Disj(C_1, C_2)$ is as a shorthand for $C_1 \sqcap C_2 \sqsubseteq \bot$.

From our experience in ontology debugging, we notice that this antipattern comes from a misunderstanding of the subClassOf relation. When the ontology developer has explicitly expressed that C_1 and C_2 are equivalent and disjoint, (s)he wants to say that C_1 and C_2 share some common properties and C_1 has more properties than C_2 (or vice versa). After a short training session the developer would discover that (s)he really wants to express that C_1 is a subclass of C_2 ($C_1 \sqsubseteq C_2$). Another possibility is that the ontology developer explicitly expressed that C_2 is a parent class of C_1. But, these two classes are determined as disjoint from each other by a reasoner.

[5] Additional information, such as examples and SPARQL queries, are available at [7].
[6] antipatterns are abstract structures that can have several DL forms.

AndIsOr (AIO) antipattern – DLAP category

$$C_3 \sqsubseteq C_1 \sqcap C_2;\ \ Disj(C_1, C_2); \tag{4}$$
$$C_3 \equiv C_1 \sqcap C_2;\ \ Disj(C_1, C_2); \tag{5}$$
$$C_3 \sqsubseteq \exists R.(C_1 \sqcap C_2);\ \ Disj(C_1, C_2); \tag{6}$$
$$C_3 \equiv \exists R.(C_1 \sqcap C_2);\ \ Disj(C_1, C_2); \tag{7}$$

This antipattern appears due to the fact that in common linguistic usage, "and" and "or" do not correspond consistently to logical conjunction and disjunction respectively [15]. An example is presented in [7].

OnlynessIsLoneliness (OIL) antipattern – DLAP category

$$C_3 \sqsubseteq \forall R.C_1; C_3 \sqsubseteq \forall R.C_2;\ \ Disj(C_1, C_2); \tag{8}$$
$$C_3 \equiv \forall R.C_1; C_3 \sqsubseteq \forall R.C_2;\ \ Disj(C_1, C_2); \tag{9}$$
$$C_3 \equiv \forall R.C_1; C_3 \equiv \forall R.C_2;\ \ Disj(C_1, C_2); \tag{10}$$

C_1 and C_2 are defined as disjoint. The ontology developer created an universal restriction to say that instances of C_3 can only be linked with property R[7] to instances of C_1. Next, a new universal restriction is added saying that instances of C_3 can only be linked with R to instances of C_2. During a long development process, the ontology developer forgot the previous axiom that can be inherited from any of the parent classes.

3 SPARQL-based Detection of Ontology Antipatterns

In this section we describe the different methods that we have elaborated in order to detect antipatterns in OWL ontologies by means of SPARQL and SPARQL-DL queries, based on the usage of the PatOMat ontology pattern detection tool [3]. This tools is part of the PatOMat suite of tools, which is focused on the detection of patterns in ontologies and their transformation. This detection tool is based on Jena 2.6.2[1] and Pellet 2.0.1[5], and enables the processing of a set of SPARQL queries over a set of ontologies, producing a report in terms of numbers of patterns detected and details for each ontology. It processes either only asserted axioms or both inferred and asserted axioms of given ontology.

Using this tool we are querying an OWL ontology by means of a query language (SPARQL) that is agnostic about the underlying knowledge representation model of OWL, i.e. we are actually querying the RDF serialization of an OWL ontology. There are also other available options in the current state of the art for OWL ontology pattern detection and transformation. First, there is OPPL language and its associated tools described by [12]. This language enables axiom-based manipulation with an OWL ontology. Second, there is a

[7] To be detectable, property R must have at least a value, normally specified as a (minimum) cardinality restriction for that class, or with existential restrictions.

language alternative for an OWL ontology querying Terp [4] which is based on the OWL Manchester syntax. While SPARQL is the language dedicated to query RDF triples, OPPL and Terp are dedicated to query the RDF serialization of OWL expressions because they contain OWL constructs like subClassOf, ComplementOf, DisjointWith. Nevertheless, in order to make queries easier (using some shortcuts for DL expressions in RDF syntax, e.g. omitting RDF collection vocabulary) we used SPARQL-DL abstract syntax defined in [17]. This enables us to express queries in more compact way. To plug in such queries into our approach we developed a query translator that transforms an input query in SPARQL-DL abstract syntax into a SPARQL query. SPARQL-DL queries and SPARQL queries are available on our antipattern web-page [7].

Transforming antipatterns into SPARQL-DL queries is not a trivial task. For each antipattern, several SPARQL-DL queries are needed to detect antipattern occurrences in OWL class definition. The difficulties come from several points:

- An antipattern can be associated to several logical formulae in DL syntax. For example, we presented 3 formulae for OIL antipatterns.
- Some logical formulae are composed of several atomic axioms. We defined an atomic axiom as a condition (necessary $\sqsubseteq$ or sufficient $\equiv$) associated to a named class C using at most one constructor ($\forall$, $\exists$, $\neg$ or $\sqcap$) and its associated operands: one class and one property for $\forall$, $\exists$ and two classes for $\sqcap$. All these classes should be named classes. An example of atomic axiom can be $C \sqsubseteq \exists R.X$. For example, the 3 formulae of the OIL antipattern contain 3 atomic axioms.
- Ontology developers can have very different implementation styles when designing an OWL ontology. For example, some developers prefer to write long class definitions. In that case, a class is defined by a conjunction of unamed classes: $C \sqsubseteq (\exists R.X) \sqcap (\forall R.Y) \sqcap \ldots$. Others can prefer to write short definitions. A class is defined by a set of atomic axioms: $C \sqsubseteq \exists R.X; C \sqsubseteq \forall R.Y; C \sqsubseteq \ldots$. Thus, in the case of an antipattern formula, an atomic axiom can be located at different places in the class definition.
- An atomic axiom can belong to the class definition or can be inherited from a parent class definition.
- An atomic axiom can be stated by the ontology developer or inferred by a reasoner.

To build our queries, we first imagine different versions of each antipattern formulae using the SPARQL-DL abstract syntax. We try to imagine where an atomic axiom can be stated by the ontology developer in a class definition. We limit our imagination to class definitions that have at most four conjunctions. We also try to imagine the different manner to express a disjoint axiom. We take in account the fact that:

- disjoint axioms are symetric $Disj(C_1, C_2) \vDash Disj(C_2, C_1)$,
- disjoint axiom can be inferred from a logical negation $C_1 \sqsubseteq \neg C_2 \vDash Disj(C_1, C_2)$.

Then we automatically translate each SPARQL-DL queries into SPARQL ones. Notice that a SPARQL-DL query is just a simplification of a SPARQL

query, which represent an exact translation of the previous one. We also automatically generate SPARQL queries which merges all the different versions.

Now we will describe four methods that we have followed in order to detect antipatterns in the ontology corpus. Overall workflow of our approach is depicted in Figure 1.

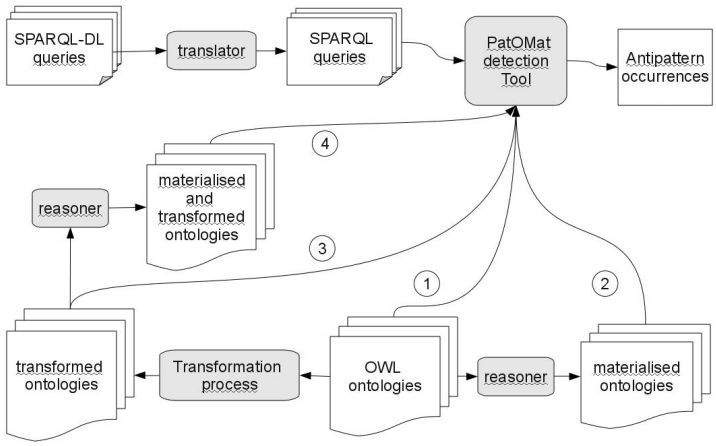


Fig. 1. The antipattern detection methods

Method 1: Use of SPARQL Queries over Asserted OWL Ontology Axioms In this approach, we take into account that SPARQL query engines per se do not consider inferences that can be done with OWL ontologies. However, our assumption is that there will be cases where ontologies cannot be processed by a reasoner or the reasoner results cannot be obtained in a reasonable time. This normally happens with large ontologies or with ontologies with a large number of errors. For example when several transitive properties are used in numerous class definitions, the reasoner reaches an out of memory alarm.

Method 2: Use of SPARQL Queries over Inferred and Asserted Ontology Axioms If it is possible to use a reasoner, we materialise all the inferences that can be done by an OWL reasoner on the ontologies and then run SPARQL queries over the resulting ontologies, called materialised ontologies.

Method 3 and 4: Use of SPARQL Queries over Transformed OWL Ontologies Due to the complexity of creating a large number of SPARQL-DL queries for an antipattern and to the fact that different ontology developers may have different

implementation styles, we propose to follow a two steps process where we apply transformations before executing the queries. Transformations have two goals: to harmonise the implementation style of the ontology and to simulate some of the axioms inferred by a reasoner. This last goal is useful only for ontologies that can not be processed by a reasoner.

The current transformations that we apply are:

- If the ontology contains $C_1 \equiv C_2$ where C_1 and C_2 are named classes, we add two new axioms $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$.
- If a named class is defined by conjunction of named or unnamed classes, we split it into several simpler axioms. E.g., considering the following class definition: $C \sqsubseteq X \sqcap Y$, in that case we add two axioms $C \sqsubseteq X$ and $C \sqsubseteq Y$.
- If a parent class contains an axiom, we add it also in its direct child class. E.g., considering the following definition of the class: $C_1 \sqsubseteq \exists R.X$. If $C_{1.1}$ is a direct child of C_1, $C_{1.1} \sqsubseteq C_1$, we add the axiom $C_{1.1} \sqsubseteq \exists R.X$. In this case, the transformation is not repeated over the class hierarchy.

In this paper, we have explored the behaviour of the SPARQL query detection method both after transformation on the asserted ontology and on the materialised ontology.

4 Experimentation: Finding Antipatterns in Real-world Ontologies

In this section, we describe the results of our experiments with a corpus of ontologies downloaded directly from the Web and by the Watson semantic search engine. We will first describe how we have built the ontology corpus, and then we present the results of applying the different methods from Section 3 over this ontology corpus.

4.1 Building a Corpus of (Debuggable) OWL Ontologies

We have used the Watson API [6] to retrieve publicly available ontologies and we have always accessed these ontologies using the Watson cache, since there are sometimes mismatches between the stored URIs of those ontologies and the actual files that can be obtained. We searched ontologies satisfying the following two constraints: they should be represented in OWL and they should have at least five classes. We collected 2927 unique ontologies. Next, we checked the consistency of all these ontologies using the Pellet reasoner, and 71 of them were classified as inconsistent.[8] From inconsistent ontologies, we removed the whole ABox so that it is possible to use a reasoner as proposed in our second method (none of our antipatterns considers the ABox, and hence the removal of the ABox

[8] We use the definition of inconsistency proposed by [18]. An ontology is inconsistent, if there is no interpretation that is a model for it. An ontology is incoherent if it contains at least one unsatisfiable class.

does not have any impact on the results obtained). This was done by OWL-API [2], which resulted in five less ontologies, since they were not parsable by this API. Consequently, the corpus is composed of 66 incoherent ontologies, that is, 66 ontologies that contain at least one unsatisfiable class.

From these ontologies we built three sets of ontologies [9]:

1. *Antipattern ontologies*: 5 ontologies in this set have already been used for the creation and update of the antipattern catalogue presented in [9]. It contains the HydrOntology (which has 159 classes whose 114 are unsatisfiables), the Forestal Ontology (which has 93 classes whose 62 are unsatisfiable), the Tambis ontology (which has 395 classes whose 112 are unsatisfiable), the Sweet Numeric ontology (which has 2364 classes whose 2 are unsatisfiable) and the University ontology of the MIND Lab (which has 29 classes whose 7 are unsatisfiable). Notice that in our experiment Hydrontology and the Tambis ontologies cannot be processed by Pellet in a reasonable time.
2. *W3C/DL ontologies*: we noticed that 31 ontologies were build by DL experts in order to test reasoner performance and results. These ontologies are characterized by having less than 18 classes (whose at most 4 classes are unsatisfiables ones). The axioms contained are very complex: inverse properties, functional properties, lots of conjunctions or disjunctions etc. But all these ontologies can be processed by Pellet.
3. *Web ontologies*: this set contains heterogeneous ontologies from various domains. There are huge ontologies which contain more than 1000 classes and Pellet cannot process them in a reasonable time, e.g. an old version of the Open CyC ontology, the Computer Science for Non-Computer Scientists ontology. There are also medium size ontologies where the number of classes is up to 100 which Pellet can process, e.g. Ontubi (an Ontology for Ubiquitous Computing) or the wine ontology.

4.2 Experiments

We made the following experiments over the 3 sets of ontologies, using the antipattern detection methods described in Section 3:

1. *SP*: a detection in the original ontologies using SPARQL[10] queries and no inference (only with asserted axioms).
2. *SP+R*: a detection in the materialised ontologies (asserted and inferred axioms) using SPARQL queries after applying a reasoner (Pellet).
3. *SP_Trans*: a transformation of the original ontologies and detection using SPARQL queries and no inference (only with asserted axioms).
4. *SP_Trans+R*: a transformation of the original ontologies and detection in the materialisation of these harmonised ontologies after applying a reasoner.

[9] All of these ontologies are available from [7].

[10] Let us note that all SPARQL queries in our experiments were automatically converted from original SPARQL-DL queries.

In some of these experiments we also use the keyword MANUAL to refer to the manual detection process using the basic debugging tools provided by ontology editors. The manual detection method is applicable only on the antipattern and the W3C/DL ontologies sets. This detection method is a baseline with respect to what can be detected using current state of the art debugging tools.

Evaluation of Antipattern Detection Precision We have evaluated the precision of the antipattern detection process. We have analysed manually each of the ontologies in our three sets and have assigned to each set one of the following three values:

- *TI* (True Inconsistency): the antipattern occurrence participates in the unsatisfiability of classes or the modelling error.
- *UI* (Unknown Inconsistency): the antipattern occurrence may be linked to the unsatisfiability of classes or modelling error, but the evaluator is not sure about it. Notice that the evaluator may find difficult to make a choice when the debuggable version of the considered ontology is not available.
- *FI* (False Inconsistency): the antipattern occurrence does not participate in the unsatisfiability of classes or modelling error.

4.3 Results

SOE detection In this case we look for a single atomic axiom written by the ontology developer. Thus only one SPARQL query is necessary to retrieve SOE occurrences and only the SP experiment was made over all 3 sets of ontologies.

set	number (nr.) of results	nr. of TI	nr. of UI	nr. of FI	nr. of onto
antipattern	16	15	0	1	2
W3C/DL	1	0	1	0	1
web	12	10	2	0	2

Table 1. SOE antipattern detection.

Due to the simplicity of the SOE antipattern, the most suitable detection method is the first method. Neither reasoner nor transformation process is needed for the detection of the SOE antipattern. The SPARQL query associated to the SOE antipattern reached 86% of precision: 25 occurrences over 29 are classified as true inconsistencies.

EID detection The EID antipattern is composed of two atomic axioms, and two formulae are possible. We defined 8 SPARQL queries associated to this antipattern. We also use 4 detection methods. Our results are limited to the fact that some ontologies cannot be processed by a reasoner in a reasonable time.

set	method	nr. of results	nr. of TI	nr. of UI	nr. of FI	nr. of onto
antipattern	manual	14	-	-	-	4
antipattern	SP	7	7	0	0	2
antipattern	SP+R	5885	14	5871	0	2
antipattern	SP_Trans	7	7	0	0	2
antipattern	SP_Trans+R	5885	14	5871	0	2
W3C/DL	manual	8	-	-	-	4
W3C/DL	SP	4	4	0	0	4
W3C/DL	SP+R	48	7	41	0	4
W3C/DL	SP_Trans	5	5	0	0	4
W3C/DL	SP_Trans+R	48	7	41	0	4
web	SP	9	9	0	0	1
web	SP+R	126	0	126	0	1
web	SP_Trans	9	9	0	0	1
web	SP_Trans+R	132	0	132	0	1

Table 2. EID antipattern detection.

Table 2 shows the results of our detection methods. We notice that using a reasoner creates some unexpected occurrences of EID antipattern. Reasoner infers that an unsatisfiable class is an equivalent to another unsatisfiable class and they are also disjoint from each others. Thus using a reasoner is not a good solution for a detection of this antipattern. The transformation improves a little bit the detection of this antipattern. It seems to be a promising direction of future work that we should improve the transformation procedure to detect more EID occurrences.

AIO detection The AIO pattern is composed of 2 atomic axioms. In Section 2 we have presented 4 formulae but in theory more formulae are possible. We imagine that a class definition can be composed of maximum 4 conjunctions: $C_3 \sqsubseteq C_w \sqcap C_x \sqcap C_y \sqcap C_z$. We defined 24 SPARQL queries corresponding to the C_1 and C_2 classes at different location of formulae. The transformation process modified the AIO pattern. Thus we added new SPARQL queries associated to the new pattern: $C_3 \sqsubseteq C_1 \sqcap C_2 \vDash C_3 \sqsubseteq C_1; C_3 \sqsubseteq C_2$.

Table 3 shows the results of our detection methods for the AIO antipattern. In this case our results are far from optimal. None of our detection method can detect the AIO occurrences in the antipattern set. This is due to the incapacity of our method to detect the atomic axiom $Disj(C_1, C_2)$ without a reasoner. Notice that the second detection method detects all the occurrences of the AIO pattern on the W3C/DL set. Thus it means that this antipattern needs a reasoner to be accurately detected.

OIL detection The OIL pattern is composed of 3 atomic axioms. We have presented 3 formulae but more formulae are possible depending on the implementation style of an ontology developer [7]. For these formulae, we imagine that a class definition can be composed of two conjunctions parts. In all, we defined 84 SPARQL queries.

set	method	nr. of results	nr. of TI	nr. of UI	nr. of FI	nr. of onto
antipattern	manual	6	-	-	-	3
antipattern	SP	1	1	0	0	1
antipattern	SP+R	3	2	1	0	2
antipattern	SP_Trans	1	1	0	0	1
antipattern	SP_Trans+R	53761	0	53761	0	2
W3C/DL	manual	9	-	-	-	8
W3C/DL	SP	2	2	0	0	2
W3C/DL	SP+R	9	9	0	0	8
W3C/DL	SP_Trans	4	2	2	0	3
W3C/DL	SP_Trans+R	236	37	199	0	8
web	SP	0	0	0	0	0
web	SP_Trans	67	0	67	0	1

Table 3. AIO antipattern detection.

set	method	nr. of results	nr. of TI	nr. of UI	nr. of FI	nr. of onto
antipattern	manual	8	-	-	-	3
antipattern	SP	2	2	0	0	2
antipattern	SP+R	2	2	1	0	2
antipattern	SP_Trans	2	2	0	0	2
antipattern	SP_Trans+R	72	6	66	0	2
web and W3C/DL		0	0	0	0	0

Table 4. OIL antipattern detection.

Results from Table 4 are surprising. In the case of the antipattern ontologies set we notice that the disjoint atomic axiom was not detected because it is not stated by the ontology developer. Furthermore using a reasoner produces unexpected antipattern occurrences. Thus any of our detection methods is good enough to detect OIL antipattern. Maybe for this specific pattern it is not necessary to detect exactly all the atomic axioms of the pattern. We should limit our detection method to the beginning of the OIL pattern without the disjoint axiom.

5 Conclusion and Future Work

In this paper we have shown how antipatterns can be detected using different methods that are based on the use of SPARQL queries, OWL reasoners and transformation tools. First, we have presented several antipatterns. Second, we have proposed different detection methods. Then, we applied them on a set of publicly available inconsistent ontologies. Finally, we have tried to figure out what are the best detection methods to be used for each antipattern. In many cases these antipattern detection methods are very sensitive to the *implemen-*

tation style of the ontology developer. Thus we recommand to avoid long class definition and to limit the use of unamed classes. Our future work will focus on the refinement of the methods that we have proposed in this paper. We will also try to design new antipatterns and detect them appropriately.

Ondřej Šváb-Zamazal has been partially supported by CSF grant no. P202/10/1825.

References

1. Apache jena. `http://jena.apache.org/`, 2012.
2. The owl api. `http://owlapi.sourceforge.net/`, 2012.
3. Patomat ontology pattern detection tool. `http://owl.vse.cz:8080/DetectionTool/`, 2012.
4. Pellet 2.1: Introducing terp. `http://weblog.clarkparsia.com/2010/04/01/pellet-21-introducing-terp`, 2012.
5. Pellet: Owl 2 reasoner for java. `http://clarkparsia.com/pellet/`, 2012.
6. Watson: Exploring the semantic web. `http://watson.kmi.open.ac.uk/WS_and_API.html`, 2012.
7. web site related to our ontology antipattern detection methods. `https://sites.google.com/site/ontologyantipattern`, 2012.
8. K. Clark. Pellint: An ontology repair tool. `http://weblog.clarkparsia.com/2008/07/02/pellint-an-ontology-repair-tool/`, 2008.
9. O. Corcho, C. Roussey, L. M. Vilches Blázquez, and I. Pérez. Pattern-based OWL ontology debugging guidelines. In Proceedings of WOP, CEUR Workshop proceedings, pages 68–82, October 2009.
10. N. Guarino and C. A. Welty. Evaluating ontological decisions with OntoClean. Commun. ACM, 45(2):61–65, 2002.
11. M. Horridge, B. Parsia, and U. Sattler. Laconic and precise justifications in OWL. In Proceedings of ISWC, pages 323–338, 2008.
12. L. Iannone, A. L. Rector, and R. Stevens. Embedding knowledge patterns into OWL. In Proceedings of ESWC, pages 218–232, 2009.
13. A. Kalyanpur, B. Parsia, E. Sirin, and J. Hendler. Debugging unsatisfiable classes in OWL ontologies. Journal of Web Semantics, 3(4):268–293, 2005.
14. V. Presutti, A. Gangemi, S. David, G.A. de Cea, M.C. Suárez-Figueroa, E. Montiel-Ponsoda, and M. Poveda. NeOn deliverable D2. 5.1. a library of ontology design patterns: reusable solutions for collaborative design of networked ontologies. 2008.
15. A. L. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL pizzas: Practical experience of teaching OWL-DL: common errors & common patterns. In Proceedings of EKAW, pages 63–81, 2004.
16. C. Roussey, O. Corcho, O. Šváb-Zamazal, F. Scharffe, and S. Bernard. Antipattern detection in web ontologies: an experiment using sparql queries. In proceedings of EGC, pages 321–326, 2012.
17. E. Sirin and B. Parsia. SPARQL-DL: SPARQL query for OWL-DL. In Proceedings of OWLED, 2007.
18. H. Stuckenschmidt. Debugging OWL ontologies - a reality check. In Proceedings of EON, 2008.
19. O. Šváb-Zamazal and V. Svátek. Analysing ontological structures through name pattern tracking. In Proceedings of EKAW, pages 213–228, 2008.

Relational Database to RDF Mapping Patterns

Juan Sequeda[1], Freddy Priyatna[2], and Boris Villazón-Terrazas[2]

[1]Department of Computer Science, The University of Texas at Austin
jsequeda@cs.utexas.edu
[2]OEG-DIA, FI, Universidad Politécnica de Madrid, Spain
{fpriyatna,bvillazon}@fi.upm.es

Abstract. In order to integrate relational databases into Semantic Web applications, relational databases need to be mapped to RDF. The W3C RDB2RDF Working Group is in the process of ratifying two standards to map relational databases to RDF: Direct Mapping and R2RML mapping language. Through our experience as implementors of two RDB2RDF systems: Ultrawrap and Morph, and as authors of R2RML mappings, we have observed mappings that are reusable in order to solve a commonly occurring problem. In this paper, we have compiled these mappings and present a non-exhaustive list of RDB2RDF Mapping Patterns. We aspire that the mapping patterns in this paper are considered as a starting point for new mapping patterns.

Key words: RDB2RDF, Mapping Patterns, Mapping Language, R2RML, Relational Databases, SPARQL, SQL

1 Introduction

In order to integrate relational databases into Semantic Web applications, relational databases need to be mapped to RDF. The W3C RDB2RDF Working Group is in the process of ratifying two standards to map relational databases to RDF: Direct Mapping[1] and R2RML (Relational Database to RDF Mapping Language)[2]. Direct Mapping is the default way of representing a relational database as RDF based on the structure of the database schema. R2RML is a language for expressing customized mappings from relational databases to RDF.

As implementors of Ultrawrap[1][8] and Morph[2], two RDB2RDF systems, that support the Direct Mapping and R2RML standards; and as authors of several R2RML mappings that have been the basis of several projects including the W3C RDB2RDF Test Cases[9], we have observed mappings that are reusable in order to solve a commonly occurring problem. In this paper, we present a series of reusable mappings, which we define as RDB2RDF Mapping Patterns. The mappings are represented in R2RML.

We would like to point out that this is not an exhaustive list of mapping patterns. The mappings patterns that we present are based on our experience.

[1]http://www.capsenta.com/
[2]https://github.com/jpcik/morph/

Assuming the RDB2RDF standards are widely adopted, we expect the mapping patterns to increase. We aspire that the mapping patterns in this paper are considered as a starting point for new mapping patterns.

2 A Motivating Example

We present an example that motivates the need of RDB2RDF mapping patterns. Due to lack of space, we do not present an overview of R2RML. We refer the reader to the R2RML spec [2]. Assume you have a table Student with attributes id, name, and homephone. An application would like to map the table person to foaf:Person, create URIs based on the attribute id, map the attribute name to foaf:name and homephone to foaf:phone. The following R2RML mapping will produce the desired output:

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{id}"; ];
  rr:predicateObjectMap [ rr:predicate foaf:name;
    rr:objectMap [rr:column "name"];
  rr:predicateObjectMap [ rr:predicate foaf:phone;
    rr:objectMap [rr:column "homephone"]; ] .
```

We observe a One to One mapping between tables and ontology classes and a One to One mapping between attributes and ontology properties.

Now assume that the table Student has a new attribute, mobilephone, which we would also like to map to foaf:phone. This means that we would need to have a Many to One mapping between attributes and an ontology property. The previous mapping could be augmented by adding rr:objectMap [rr:column "mobilephone"]; to the existing rr:predicateObjectMap that has foaf:phone as a predicate. Another solution would be to repeat the entire rr:predicateObjectMap, but with a rr:column of mobilephone. This type of pattern impacts query performance. The following SPARQL query: `SELECT ?s ?o WHERE {?s foaf:phone ?o}` would get translated to the following SQL query: `SELECT id, homephone FROM Student UNION SELECT id, mobilephone FROM Student`. If we were to increase the amount of attributes mapped to the same ontology property, the size of the SQL query would increase. This example suggests that there is a tradeoff between mapping patterns and query performance. In order to further study tradeoffs and design decisions of RDB2RDF mappings, it is important to understand the different types of mapping patterns. In the next section, we present fourteen mapping patterns which we have observed as implementors of RDB2RDF systems and authors of R2RML mappings.

3 R2RML Mapping Patterns

A RDB2RDF mapping pattern is a reusable mapping that solves a commonly occurring problem. We present four type of mapping patterns: Attribute Mapping Patterns, Table Mapping Patterns, Join Mapping Patterns and Value Translation Patterns. Each pattern consists of a name, a question that defines the problem that is being addressed, description of the context, description of the solution in R2RML, an example R2RML mapping, a discussion and related patterns. Some mapping patterns may consist of different R2RML solutions.

3.1 Table Mapping Patterns

Tables in a relational database are (usually) mapped to ontology classes[3]. Each record of the table is mapped to an instance of the ontology class. In R2RML, every TripleMap must have exactly one rr:logicalTable and one rr:subjectMap. The rr:logicalTable defines the table (or SQL query) that is being mapped. The rr:subjectMap defines how to generate the subjects for the RDF triples. The following patterns define different ways that a table can be mapped to an ontology class. Patterns to generate the URIs for the subjects are out of the scope of this work. We refer the reader to [3].

Pattern 1: One to One Table Mapping

How to map a table to an ontology class?
Context: An application would like to map a tables to an ontology class. Moreover, every record of the tables is mapped to an instance of the corresponding ontology classes. For example, the table student is mapped to foaf:Person.
Solution: Create a TriplesMap for the table and specify the rr:logicalTable whose value corresponds to the table name. In the TriplesMap, create a rr:subjectMap with a rr:template to define the URI template for each row. Finally, the rr:subjectMap will have a rr:class corresponding to the ontology class for that table.
Example R2RML Mapping

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
    rr:subjectMap [ rr:class foaf:Person;
    rr:template "http://example.com/resource/Student/{sid}" ] .
```

Discussion: This is the simplest pattern for table mapping. This is the case of the Direct Mapping, which automatically generates a unique ontology class for each table. However, a user has the option to specify a particular ontology class.
Related Patterns: N/A

Pattern 2: One to Many Table Mapping

How to map a table to several ontology classes?
Context: An application would like to map a table to many ontology classes. Moreover, every record of the table is mapped to an instance of the corresponding ontology classes. For example, the table student is mapped to foaf:Person and ex:Student.
Solution: Create a TriplesMap for the table and specify the rr:logicalTable whose value corresponds to the table name. In the TriplesMap, create a rr:subjectMap with a rr:template to define the URI template for each row. Finally, the rr:subjectMap will have multiple rr:class that correspond to the ontology classes for that table.
Example R2RML Mapping

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template
    "http://example.com/resource/Student/{sid}";
    rr:class foaf:Person; rr:class ex:Student ].
```

Discussion: This pattern extends Pattern 1 by adding multiple rr:class.
Related Patterns: Pattern 1

[3]Except for the case when the table represents a many-to-many relationship

Pattern 3: Many to One Table Mapping

How to map several tables to an ontology class?

Context: An application would like to map many tables to an ontology class. Moreover, every record of these table are mapped to an instance of the corresponding ontology class. For example, the tables student and professor are both mapped to foaf:Person.

Solution: Repeat the solution in pattern 1 for each table to be mapped.

Example R2RML Mapping

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:class foaf:Person;
    rr:template "http://example.com/resource/Student/{sid}" ] .

<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "professor" ];
  rr:subjectMap [ rr:class foaf:Person;
    rr:template "http://example.com/resource/Professor/{sid}" ] .
```

Discussion: This pattern extends Pattern 1 and it is used when instances of an ontology class may be distributed over several tables.

Related Patterns: Pattern 1

Pattern 4: Many to Many Table Mapping

How to map several tables to several ontology classes?

Context: An application would like to map a table to several ontology classes. Additionally, the application would like to map an ontology class to several tables.

Solution: Repeat solution of pattern 2 for each table to be mapped.

Example R2RML Mapping

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}";
                  rr:class foaf:Person; rr:class ex:Academic ] .

<TriplesMapProf> a rr:TriplesMap; rr:logicalTable [ rr:tableName "professor" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Professor/{sid}";
                  rr:class foaf:Person; rr:class ex:Academic ] .
```

Discussion: This pattern extends Pattern 3. In addition to instances of an ontology classes are distributed over several database tables, each row of the database table produces multiple ontology instances.

Related Patterns: Pattern 2, Pattern 3

3.2 Attribute Mapping Patterns

Attributes of tables are mapped to ontology properties. In R2RML, a TripleMap can have zero or more rr:predicteObjectMap, which in turn specifies a predicate-object pair. The following patterns define different ways that an attribute can be mapped to an ontology property.

Pattern 5: One to One Attribute Mapping

How to map an attribute to an ontology property?
Context: An application would like to map an attribute to an ontology property. For example, the attribute firstname is mapped to foaf:givenName.
Solution: Given a TripleMap, create a rr:predicateObjectMap for the attribute, which has only one rr:predicate for the ontology property and a rr:objectMap for the attribute.
Example R2RML Mapping

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [
    rr:template "http://example.com/resource/Student/{sid}"; ];
  rr:predicateObjectMap [ rr:predicate foaf:givenName;
    rr:objectMap [rr:column "firstname"]; ] .
```

Discussion: This is the simplest pattern for attribute mapping. This is the case of the Direct Mapping, which automatically generates a unique ontology property for each attribute.
Related Patterns: N/A

Pattern 6: One to Many Attribute Mapping

How to map an attribute to several ontology properties?
Context: An application would like to map an attribute to several ontology properties. For example, the attribute lastname is mapped to foaf:familyName and ex:apellido.
Solution 1: Given a TripleMap, create a rr:predicateObjectMap for the attribute, which has a rr:predicate for each ontology property and a rr:objectMap for the attribute.
Example R2RML Mapping for Solution 1

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}"; ];
  rr:predicateObjectMap [ rr:predicate foaf:familyName; rr:predicate ex:apellido;
    rr:objectMap [rr:column "lastname"]; ]
```

Solution 2: Repeat the solution for Pattern 5 for the attribute to be mapped but having the rr:predicate specific to each ontology property
Example R2RML Mapping for Solution 2

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}"; ];
  rr:predicateObjectMap [
    rr:predicate foaf:familyName; rr:objectMap [rr:column "lastname"]; ];
  rr:predicateObjectMap [
    rr:predicate ex:apellido; rr:objectMap [rr:column "lastname"]; ].
```

Discussion: Solution 1 is a short cut for Solution 2.
Related Patterns: Pattern 5

Pattern 7: Many to One Attribute Mapping

How to map several attributes to an ontology property?
Context: An application would like to map several attributes to an ontology property. For example, the attribute homephone and mobilephone is mapped to foaf:phone.

Solution 1: Given a TripleMap, create a rr:predicateObjectMap, which has only a rr:predicate for the ontological property and a rr:objectMap for each attribute.
Example R2RML Mapping for Solution 1

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}"; ];
  rr:predicateObjectMap [ rr:predicate foaf:phone;
    rr:objectMap [rr:column "homephone"]; rr:objectMap [rr:column "mobilephone"]; ].
```

Solution 2: Repeat the solution for Pattern 5 for each attribute to be mapped
Example R2RML Mapping for Solution 2

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}"; ];
  rr:predicateObjectMap [ rr:predicate foaf:phone;
    rr:objectMap [rr:column "homephone"]; ];
  rr:predicateObjectMap [ rr:predicate foaf:phone;
    rr:objectMap [rr:column "mobilephone"]; ].
```

Discussion: Solution 1 is a short cut for Solution 2. As described in the motivating example of Section 2, this pattern impacts performance on queries that select on the ontology property.
Related Patterns: Pattern 5

Pattern 8: Many to Many Attribute Mapping

How to map several attributes to several ontology properties?
Context: An application would like to map an attribute to several ontology predicates. Additionally, the application would like to map an ontology predicate to several attributes.
Solution 1: Combine Solution 1 of Pattern 6 with Solution 1 of Pattern 7.
Example R2RML Mapping for Solution 1

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}"; ];
  rr:predicateObjectMap [ rr:predicate foaf:phone; rr:predicate ex:telefono;
    rr:objectMap [rr:column "homephone"]; rr:objectMap [rr:column "mobilephone"]; ].
```

Solution 2: Combine Solution 2 of Pattern 6 with Solution 2 of Pattern 7.
Example R2RML Mapping for Solution 2

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}" ];
  rr:predicateObjectMap [ rr:predicate foaf:phone;
    rr:objectMap [rr:column "homephone"]; ];
  rr:predicateObjectMap [ rr:predicate foaf:telefono;
    rr:objectMap [rr:column "homephone"]; ];
  rr:predicateObjectMap [ rr:predicate foaf:phone;
    rr:objectMap [rr:column "mobilephone"]; ];
  rr:predicateObjectMap [ rr:predicate foaf:telefono;
    rr:objectMap [rr:column "mobilephone"]; ].
```

Discussion: Solution 1 is a shortcut for Solution 2.
Related Patterns: Pattern 6 and Pattern 7.

Pattern 9: Concatenate Attributes

How to concatenate attributes and map it to an ontology property?
Context: An application would like to concatenate several attributes and map the result to an ontology property. For example, concatenate the attributes first-name and lastname and map it to foaf:name.

Solution 1: Given a TripleMap, create a rr:predicateObjectMap which has a rr:predicate for the ontology property and the rr:objectMap as a rr:template. The concatenation is represented as a template.

Example R2RML Mapping for Solution 1

```
<TriplesMapStudent1> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}" ];
  rr:predicateObjectMap [
    rr:predicate foaf:name; rr:objectMap [ rr:template "{firstname} {lastname}" ]; ].
```

Solution 2: Create a new TripleMap with an R2RML view which consists of a rr:logicalTable that has a rr:sqlQuery which includes the concatenation. Additionally, create a rr:predicateObjectMap which has a rr:predicate for the ontology property and the rr:objectMap for the attribute which represents the concatenation in the SQL query.

Example R2RML Mapping for Solution 2

```
<TriplesMapStudent1> a rr:TriplesMap; rr:logicalTable [
    rr:sqlQuery "SELECT sid, firstname || ' ' || lastname AS fullname FROM student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}" ];
  rr:predicateObjectMap [
    rr:predicate foaf:name; rr:objectMap [rr:column "fullname"]; ].
```

Discussion: Consider the SPARQL query `SELECT ?s ?fullname WHERE {?s foaf:name ?fullname }`. Solution 1 would produce a SQL query and concat the name in the SELECT clause. Solution 2 does the concatenation operation as a SQL query as specified in rr:sqlQuery and then use this query as a subquery.

```
Solution 1 SQL : SELECT sid, firstname || '' || lastname as fullname FROM student
Solution 2 SQL : SELECT sid, fullname
                 FROM (SELECT sid, firstname || ' ' || lastname AS fullname
                       FROM student)
```

Note that these queries are semantically equivalent. From a query performance perspective, they should be equal unless the RDBMS does not have optimizations for subqueries.

Related Patterns: Pattern 5

3.3 Join Mapping Patterns

Foreign Key relationships among tables can be mapped to ontology properties. The following patterns define different ways that foreign key relationships can be mapped to an ontology property.

Pattern 10: Foreign Key between Two Tables

How to represent the relationship between two tables?

Context: An application would like to map a table to an ontology class. However, some of the property values are stored in another table. Therefore, it is necessary to perform a join to get those values.

Solution: Given two tables, one table will be considered the child and the other the parent. Create a TripleMap for each table. Given the child TripleMap, create a rr:predicateObjectMap which will have, in addition to the rr:predicate, a rr:objectMap which has a rr:parentTripleMap and a rr:joinCondition. The rr:parentTripleMap will point to the parent TripleMap and the rr:joinCondition

will have a rr:child and rr:parent which represent the join attributes in the child and parent table respectively.

Example R2RML Mapping

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.org/resource/Student/{sid}"; ];
  rr:predicateObjectMap [ rr:predicate ex:countryOfBirth;
    rr:objectMap [ rr:parentTriplesMap <TriplesMapCountry>;
      rr:joinCondition [ rr:child "country_of_birth" ;rr:parent "cid" ; ];];].

<TriplesMapCountry> a rr:TriplesMap; rr:logicalTable [ rr:tableName "country" ];
  rr:subjectMap [ rr:template "http://example.org/resource/Country/{cid}"; ].
```

Discussion: This pattern describes an R2RML mapping that joins two tables. This mapping can also be represented using Pattern 11. However, if a join involves more than two tables, then Pattern 11 must be used. The addition of another property that involves a join in Pattern 10 means that the user has to specify a new parent TriplesMap and then refer this parent TriplesMap in a rr:objectMap for the new property. On the other hand, the rr:logicalTable value stays the same and no changes needed.

Related Patterns: Pattern 11

Pattern 11: Foreign Keys between Two or more Tables

How to represent the relationship between two or more tables?

Context: An application would like to map a table to an ontology class. However, some of the property values are stored in other tables. Therefore, it is necessary to perform a joins to get those values.

Solution: Create a TripleMap with an R2RML view which consists of a rr:logicalTable that has a rr:sqlQuery which includes a SQL query that represent explictly the join(s).

Example R2RML Mapping

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:sqlQuery """
    SELECT s.sid AS sid, c.country_code AS country_code FROM student s, country c
    WHERE s.country_of_birth = c.country_id """];
  rr:subjectMap [ rr:template "http://example.org/resource/Student/{sid}"; ];
  rr:predicateObjectMap [ rr:predicate ex:countryOfBirth;
    rr:objectMap [ rr:template "http://example.org/resource/Country/{cid}"; ]; ] .
```

Discussion: If the join is between two tables, then Pattern 10 can be used. However, if the join is between more than two tables, then Pattern 10 can not be used and the SQL query must be made explicit. Unlike Pattern 10, Pattern 11 does not require to have additional TriplesMap needed in order to map the property that needs a join. An addition of a new property that involves a join does not require to create additional TriplesMap instance, but the user has to modify the SQL query.

Related Patterns: Pattern 10

Pattern 12: Many to Many Table

How to map a table that represents a many-to-many relationship between two other tables to an ontology property?

Context: A many-to-many table represents a relationship between two entities. For example, the table StudentSport records the relationship of which Students

play a specific Sport. Several students can play a sport and several sports can be played by a student. An application would like to map the many-to-many table to an ontology property.
Solution: Create a TriplesMap for the many-to-many table. Specify the rr:logicalTable whose value corresponds to the table name of the many-to many table. In the TriplesMap, create a rr:subjectMap with a rr:template to define the URI template for one of the tables of the many-to-many relationship. Create an instance of rr:predicateObjectMap which has a rr:predicate for the ontology property. Finally, create a rr:objectMap with a rr:template to define the URI template for the other table of the many-to-many relationship.
Example R2RML Mapping

```
<TriplesMapStudentSport> a rr:TriplesMap;
    rr:logicalTable [ rr:tableName "StudentSport" ];
    rr:subjectMap [ rr:template "http://example.org/resource/Student/{studentid}"; ];
    rr:predicateObjectMap [ rr:predicate ex:plays;
    rr:objectMap [ rr:template "http://example.org/resource/Sport/{sportid}" ]; ] .
```

Discussion: This mapping can also be represented through Pattern 11 given that it consists of a join between three tables (Student, StudentSport and Sport).
Related Patterns: Pattern 11

3.4 Value Translation Patterns

It is common that specific values in the database are code values which need to be translated to URIs. R2RML relies on SQL's `CASE` statement for the translation. The following patterns define different ways that values can be translated.

Pattern 13: Translate Values

How to map values in a table to URIs?
Context: An application would like to map values in the data to IRIs. For example, if the value in a job attribute is engineer, then a special IRI needs to be generated. A translation using rr:template is not possible because templates can only use the same values from the database.
Solution: Create a new TripleMap with an R2RML view which consists of a rr:logicalTable that has a rr:sqlQuery. Represents the translation in the SQL query by using SQL CASE statement.
Example R2RML Mapping

```
<#TriplesMap1> a rr:TriplesMap; rr:logicalTable [ rr:sqlQuery """
    SELECT EMP.*, (CASE JOB
        WHEN 'CLERK' THEN 'general-office' WHEN 'NIGHTGUARD' THEN 'security'
        WHEN 'ENGINEER' THEN 'engineering' END) ROLE FROM EMP  """ ];
  rr:subjectMap [ rr:template "http://data.example.com/employee/{EMPNO}"; ];
  rr:predicateObjectMap [ rr:predicate ex:role;
    rr:objectMap [ rr:template "http://data.example.com/roles/{ROLE}";
        rr:termType rr:IRI; ]; ] .
```

Discussion: The R2RML language does not have the expressivity to represent such value translation. Therefore, translating values has been pushed into SQL using the CASE statement. Query performance depends on the optimizations that a RDBMS has for the CASE statement.
Related Patterns: N/A

Pattern 14: Translate Values between Tables

How to map values in a referenced table

Context: An application would like to map a table to an ontology class. The table has a foreign key that references another table. The referenced table contains columns whose values need to be translated into URIs.

Solution 1: Combine Pattern 11 and Pattern 13.

Example R2RML Mapping for Solution 1

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:sqlQuery """
    SELECT s.sid AS sid, a.aid AS aid, (CASE a.article_type
        WHEN 'ppr' THEN 'Paper' WHEN 'ths' THEN 'Thesis') AS ArticleType
    FROM student s, article a WHERE s.sid=a.author """ ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}"; ];
  rr:predicateObjectMap [ rr:predicate ex:isAuthorOf;
    rr:objectMap [ rr:template "http://example.com/resource/{ArticleType}/{aid}";];];].
```

Solution 2: Use Pattern 10

Example R2RML Mapping for Solution 2

```
<TriplesMapStudent> a rr:TriplesMap; rr:logicalTable [ rr:tableName "student" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Student/{sid}"; ];
  rr:predicateObjectMap [ rr:predicate ex:isAuthorOf; rr:objectMap [
    rr:parentTriplesMap <TriplesMapPaper>;
    rr:joinCondition [ rr:child "sid" ; rr:parent "author" ; ]; ]; ];
  rr:predicateObjectMap [ rr:predicate ex:isAuthorOf; rr:objectMap [
    rr:parentTriplesMap <TriplesMapThesis>;
    rr:joinCondition [ rr:child "sid" ; rr:parent "author" ; ]; ]; ].

<TriplesMapPaper> a rr:TriplesMap; rr:logicalTable [
    rr:sqlQuery """SELECT author, aid FROM article WHERE article_type='ppr'""" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Paper/{aid}"; ].

<TriplesMapThesis> a rr:TriplesMap; rr:logicalTable [
    rr:sqlQuery """SELECT author, aid FROM article WHERE article_type='ths'""" ];
  rr:subjectMap [ rr:template "http://example.com/resource/Thesis/{aid}"; ].
```

Discussion: The selection of the possible solutions affect the way a user add a new article type, and also the SQL needed to execute the mappings.

In solution 1, to add a new article type, a user just need to modify the SQL query, appending the corresponding WHEN THEN pair statement. In solution 2, a new (parent) TriplesMap instance is needed for each article type. Then, in the child TriplesMap, every article type needs a predicateObjectMap property.

Now consider the following SPARQL query:
`SELECT ?s ?o WHERE { ?s ex:isAuthorOf ?o }`. The resulting SQL query for Solution 1 is the following:

```
SELECT sid, ArticleType, aid FROM (
    SELECT s.sid AS sid, a.aid AS aid, (CASE a.article_type
        WHEN 'ppr' THEN 'Paper' WHEN 'ths' THEN 'Thesis') AS ArticleType
    FROM student s, article a WHERE s.sid=a.author)
```

The resulting SQL query for Solution 2 is the following:

```
SELECT sid, aid
    FROM student S, (SELECT author, aid FROM article WHERE article_type='ppr') P
    WHERE S.sid = P.author UNION
    SELECT sid, aid
    FROM student S, (SELECT author, aid FROM article WHERE article_type='ths') T
    WHERE S.sid = T.author
```

Note that the generated SQL queries are very different and may have an impact on query performance depending on the RDBMS.

Related Patterns: Pattern 10, Pattern 11, Pattern 13

4 Related Work

In the (Object-Oriented) software community, patterns are used to describe software design structures that can be used over and over again in different systems. They provide a general solution that has to be applied in a particular context, in which the design considerations serve to decide whether the pattern is useful and how it could be implemented best [4]. A kind of software patterns are the re-engineering software patterns [6]. These patterns describe how to change a legacy system into a new, refactored system that fits current conditions and requirements. Their main goal is to offer a solution for re-engineering problems. They are also on a specific level of abstraction, that describes a process of re-engineering without proposing a complete methodology, and sometimes can suggest which type of tool to use. Therefore RDB2RDF mappings can be seen as re-engineering patterns because they map a legacy system (RDBMS) into a new system (RDF).

In the Semantic Web community, the Ontology Design Pattern portal[4] has been created in order to help in the design and quality of ontologies. Additionally, Dodds and Davis presents a pattern catalogue for modeling, publishing and consuming Linked Data [3]. These Linked Data patterns do not include RDB2RDF mapping patterns. Therefore our work is complemented by the Linked Data patterns of Dodds and Davis. Furthermore, Hert et al. presents a comparison of the expressivity of RDB2RDF mapping languages [5]. A framework for comparison is introduced, which consists of a set of fifteen features such as support for different types of mappings, datatypes, named graphs, blank nodes etc. R2RML supports all but one feature (write access) . Moreover, Rivero et al. presents fifteen RDF to RDF mapping patterns [7]. Some of these patterns are specific to RDF to RDF mappings such as Remove Language Tag while others are not applicable to RDB2RDF mappings such as Rename Class or Rename Property.

5 Conclusions and Future Work

In this paper, we have introduced fourteen RDB2RDF mapping patterns[5], which we have observed as reusable mapping throughout our experience as RDB2RDF system developers and R2RML mapping authors. As previously mentioned, this is a non-exhaustive list of mapping patterns and we aspire that the mapping patterns presented in this paper serve as a starting point. We foresee new mapping patterns in areas such as Named Graphs, Blank Nodes for anonymous or sensitive data, Metadata, Languages, Datatypes. We hope that this work encourages the Semantic Web community to further extend the RDB2RDF mapping patterns.

In certain patterns, we have identified two R2RML mapping solutions. These solutions may or may not have performance issues. As future work, we will

[4] http://ontologydesignpatterns.org/

[5] The mappings are stored at `http://mappingpedia.linkeddata.es/pattern/DataPatterns/`

thoroughly study the tradeoff between mapping patterns and query performance. Additionally, we will investigate the overlap between ontology design patterns, linked data patterns, the feature set of Hert et al. and the RDF to RDF mapping patterns of Rivero et al. with RDB2RDF mappings in general.

Finally, we have to see how to align the proposed RDB2RDF mapping patterns with Re-engineering Patterns category in the ODP Portal.

Acknowledgments: This work has been supported by the PlanetData, BabelData, and myBigData projects. Juan F. Sequeda was supported by the NSF Graduate Research Fellowship. We would like to kindly thank all W3C RDB2RDF Working Group members. We thank Richard Cyganiak for discussions.

References

1. M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda. Direct mapping of relational data to RDF. W3C Working Draft 29 May 2012, `http://www.w3.org/TR/2012/WD-rdb-direct-mapping-20120529/`.
2. S. Das, S. Sundara, and R. Cyganiak. R2rml: Rdb to rdf mapping language. W3C Working Draft 29 May 2012, `http://www.w3.org/TR/2012/WD-r2rml-20120529/`.
3. L. Dodds and I. Davis. Linked data patterns-a pattern catalogue for modelling, publishing, and consuming linked data. http://patterns.dataincubator.org/book/, 2011.
4. H. Edwards, R. Puckett, and A. Jolly. Analyzing Communication Patterns in Software Engineering Projects. In *Software Engineering Research and Practice*, pages 310–315, 2006.
5. M. Hert, G. Reif, and H. C. Gall. A comparison of rdb-to-rdf mapping languages. In *Proceedings of the 7th International Conference on Semantic Systems*, 2011.
6. R. Pooley and P. Stevens. Software Reengineering Patterns. Technical report, 1998.
7. C. R. Rivero, A. Schultz, C. Bizer, and D. Ruiz. Benchmarking the performance of linked data translation systems. In *Proceedings of the 5th Linked Data on the Web Workshop (LDOW)*, 2012.
8. J. F. Sequeda and D. P. Miranker. Ultrawrap: Sparql execution on relational data. Technical Report TR-12-10, The University of Texas at Austin, Department of Computer Sciences, 2012.
9. B. Villazón-Terrazas and M. Hausenblas. R2rml and direct mapping test cases. W3C Editor's Draft 24 July 2012, `http://www.w3.org/2001/sw/rdb2rdf/test-cases/`.

Building Ontologies by using Re-engineering Patterns and R2RML Mappings

Freddy Priyatna[1] and Boris Villazón-Terrazas[1]

[1]OEG-DIA, FI, Universidad Politécnica de Madrid, Spain
fpriyatna@delicias.dia.fi.upm.es,bvillazon@fi.upm.es

Abstract. The ontologization of non-ontological resources has led to the design of several specific methods, techniques and tools. Among those techniques, we have the Re-engineering Patterns. More specifically, we have the Patterns for re-engineering NORs (PR-NOR) that define a procedure that transforms the NOR terms into ontology representational primitives. Currently, the W3C RDB2RDF Working Group is at the final stage of formalizing R2RML, a language for describing mappings among RDB elements and RDF. In this paper we claim that it is possible to combine PR-NORs with R2RML mappings for building ontologies from relational database content, i.e., transforming the database content into an ontology schema by using Re-engineering Patterns and R2RML mappings.

Key words: Re-engineering patterns, RDB2RDF, R2RML, Ontologies

1 Introduction

During the last decade, specific methods, techniques and tools were proposed for building ontologies from existing knowledge resources. When we are transforming non-ontological resources (NORs) [4] into ontologies, the transformation process may follow one of the following approaches: (1) ***ABox transformation*** [4], which transforms the resource schema into an ontology schema, and the resource content, into ontology instances; (2) ***TBox transformation*** [4], which transforms the resource content into an ontology schema; or (3) ***Population***, which transforms the resource content into instances of an available ontology. The ABox transformation leaves the informal semantics of the transformed resources mostly untouched, whereas, the TBox transformation tries to enforce a formal semantics into them. Figure. 1 depicts the three types of transformation.

According to the survey described in [9], most of the available methods and tools deal with ABox transformation and Population. However there are some cases when it is useful to follow the TBox transformation [4], for example when we have a taxonomy stored in a particular NOR. The ontologization of non-ontological resources has led to the design of several specific methods, techniques and tools [4]. Among those techniques, we have the Re-engineering Patterns,

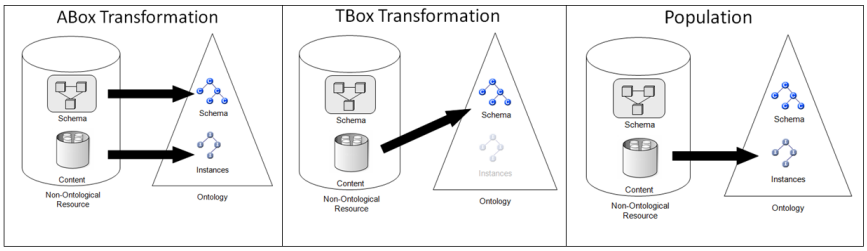


Fig. 1. NOR Transformation into ontologies

within the context of the ODP[1]. More specifically, we have the Patterns for re-engineering NORs (PR-NOR) that define a procedure that transforms the NOR terms into ontology representational primitives. Nevertheless, PR-NORs do not consider the implementation of the NOR, they just provide a general algorithm for the transformation.

The majority of non-ontological resources underpinning the Web are implementing in Relational Databases (RDB) [6]. RDB systems host a vast amount of structured data in relational tables augmented with integrity constraints [5]. When we are transforming RDB content into ontologies, we can follow two approaches (1) procedural/imperative approach, or (2) declarative approach, by defining mappings between RDB and ontology elements. There are several RDB2RDF mapping languages for describing transformation among RDB elements and ontologies [9, 7]. The RDB2RDF working group[2] is at the final stage of formalizing R2RML[3], a standard language for expressing mappings from relational databases to RDF datasets.

As we mentioned above, we provided a general algorithm to do the transformation for each of PR-NORs. However, in order to actually do the transformation, a user has to implement this algorithm in his choice of programming language (Java, Scala, etc). On the other hand, we observed that although R2RML mappings are normally used to generate ontology instances from database content (Population transformation), we figured out that when the database content follows specific patterns such as PR-NORs, then R2RML mappings can be useful in this situation. In this paper we propose to combine PR-NORs with R2RML mappings for building ontologies from relational database content, i.e., transforming the database content into an ontology schema by using Re-engineering Patterns and R2RML mappings.

The rest of the paper is organized as follows. Section 2 provides the background knowledge, by describing PR-NORs and R2RML. Then, Section 3 presents how we combine the PR-NORs with R2RML mappings for building the ontolo-

[1] `http://ontologydesignpatterns.org`
[2] `http://www.w3.org/2001/sw/rdb2rdf/`
[3] `http://www.w3.org/TR/r2rml/`

gies from relational database content, and includes two examples. Finally, Section 4 presents the conclusions and future work.

2 Background Knowledge

In this section we provide a brief description of the PR-NORs and R2RML.

2.1 Patterns for Re-engineering Non-Ontological Resources

The Patterns for Re-engineering Non-Ontological Resources (PR-NOR) [12] define a procedure that transforms the NOR terms into ontology elements. The patterns describe the transformation of classification schemes, thesauri, and lexicons into ontologies. The patterns rely on the data model[4] of the NORs. The patterns define, for every data model of the NORs, a process (expressed as an algorithm) with a well-defined sequence of activities in order to extract the NORs terms, and then to map these terms to a conceptual model of an ontology. Nevertheless, the patterns do not consider the implementation of the NOR, they just provide a general procedure for the transformation. It is worth noting that these patterns are included in the ODP Portal[5]. Table 1 lists the set of PR-NORs that perform the TBox transformation approach.

Table 1. Set of patterns for re-engineering NORs that perform the TBox transformation approach.

N	Identifier	Type of NOR	NOR Data Model	Target
1	PR-NOR-CLTX-01	Classification Scheme	Path Enumeration	Ontology Schema (TBox)
2	PR-NOR-CLTX-02	Classification Scheme	Adjacency List	Ontology Schema (TBox)
3	PR-NOR-CLTX-03	Classification Scheme	Snowflake	Ontology Schema (TBox)
4	PR-NOR-CLTX-04	Classification Scheme	Flattened	Ontology Schema (TBox)
5	PR-NOR-TSTX-01	Thesaurus	Record-based	Ontology Schema (TBox)
6	PR-NOR-TSTX-02	Thesaurus	Relation-based	Ontology Schema (TBox)
7	PR-NOR-LXTX-01	Lexicon	Record-based	Ontology Schema (TBox)
8	PR-NOR-LXTX-02	Lexicon	Relation-based	Ontology Schema (TBox)

In a nutshell, the main fields of a PR-NOR are:

[4]The data model[12] is the abstract model that describes how data is represented and accessed. The data model can be different even for the same type of non-ontological resource.

[5]`http://ontologydesignpatterns.org/wiki/Submissions:ReengineeringODPs`

– Name of the pattern
– Identifier of the pattern
– Use case, description in natural language of the re-engineering problem addressed by the pattern
– Input, description in natural language of the NOR, and its graphical representation
– Output, description in natural language of the ontology created after applying the pattern, and its graphical representation
– Process, algorithm for the re-engineering process

2.2 R2RML

R2RML[6] is a language for expressing mappings from relational databases to RDF datasets. These mappings provide the ability to view existing relational data in the RDF data model, expressed in a target ontology. The input to an R2RML mapping is a relational database. The output is an RDF dataset that uses predicates and types from the target ontology. It is worth mentioning that R2RML mappings are themselves expressed as RDF graphs and written down in Turtle syntax [2]. Figure 2 shows the elements of R2RML language

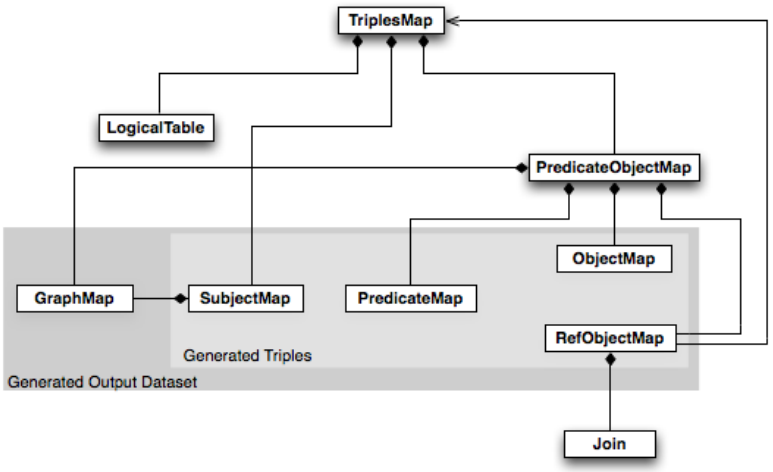


Fig. 2. An overview of R2RML

In a nutshell, an R2RML mapping points to logical tables to get data from the database. A logical table can be (1) a database table, (2) a view, or (3) a valid SQL query. Each logical table is mapped to RDF using a triples map. A

[6] `http://www.w3.org/TR/r2rml/`

triples map defines rules that map each row in the logical table to a set of RDF triples. Those rules have two main parts (1) a subject map, which generates the subject of all RDF triples that will be created from a logical table row; and (2) multiple predicate-object maps that consist of predicate maps and object maps (or referencing object maps). Triples are produced by combining the subject map with a predicate map and object map, and applying these three to each logical table row. It is possible that a triples map can contain graph maps that place some or all of the triples into named graphs, but by default, all RDF triples are in the default graph of the output dataset.

Next, we present a basic example in order to illustrate how to specify R2RML mappings[7]. Let us consider the database depicted in Figure 3. This database contains a table, one primary key, two columns, and one row.

Student

ID (PK) INTEGER	Name VARCHAR(50)
10	Venus

Fig. 3. Example database

For the that database we have the following R2RML mapping

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: <http://example.com/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@base <http://example.com/base/> .

<TriplesMap1> a rr:TriplesMap;
rr:logicalTable [ rr:tableName "Student" ];
rr:subjectMap [ rr:class foaf:Person;
      rr:template "http://example.com/Student/{ID}/{Name}"; ];
rr:predicateObjectMap [ rr:predicate      ex:id ;
      rr:objectMap  [ rr:column "ID";  ] ];
rr:predicateObjectMap [  rr:predicate  foaf:name ;
      rr:objectMap  [ rr:column "Name" ] ].
```

Finally, it is worth mentioning that neither the RDB2RDF Use Cases and Requirements [8] nor the R2RML mappings in the R2RML Test Cases document[9] provide or say anything about generating ontologies from database content.

[7] Please refer to the R2RML specification and its testcases to check a detailed list of R2RML mapping examples.

[8] http://www.w3.org/TR/rdb2rdf-ucr/

[9] http://www.w3.org/2001/sw/rdb2rdf/test-cases/

3 Combining PR-NORs and R2RML mappings

In this section we present how to generate ontologies from database content by using PR-NORs and R2RML mappings. Our combination of PR-NORs and R2MRL mappings is two-fold, first (1) we want to specialize the PR-NORs that perform a TBox transformation by specifying Relational databases as NOR, (2) we want to show that it is possible to generate ontology schema triples using R2RML mappings.

Using R2RML mappings for transforming PR-NOR patterns brings two benefits. The first benefit is since R2RML mappings are expressed in RDF, we can store and reuse them. The second is, there are already several R2RML engines[10], therefore, it will be possible to execute the mappings and generate the ontologies in a short time.

Using our approach, each PR-NOR has the corresponding R2RML mapping. The mappings will be executed by an R2RML engine and the result of that execution will generate an ontology represented as an RDF document.

All the patterns, mappings, and some other files (sql dump, result in graphical representation etc) are available here at:

- Pattern Description : mappingpedia:pattern/SchemaPatterns/{PatternID}
- Mapping Algorithm : mappingpedia:pattern/SchemaPatterns/{PatternID}/algorithm.txt
- Mapping Example : mappingpedia:pattern/SchemaPatterns/{PatternID}/mapping-example.ttl
- SQL General : mappingpedia:pattern/SchemaPatterns/{PatternID}/sql-general.sql
- SQL Example : mappingpedia:pattern/SchemaPatterns/{PatternID}/sql-example.sql
- RDF Result Example : mappingpedia:pattern/SchemaPatterns/{PatternID}/result-example.nt

Where mappingpedia represents `http://mappingpedia.linkeddata.es/`

Next, we show examples of how to build R2RML mappings that correspond to the PN-NORs patterns. We use snowflake model as the representation of classification schema pattern and term-based relational model as the representation of thesaurus pattern.

PR-NOR-CLTX-03 : Pattern for re-engineering a classification scheme following the snowflake data model into an ontology schema A classification scheme is a rooted tree of terms, in which each term groups entities by some particular degree of similarity. The semantics of the hierarchical relation between parent and children terms may vary depending on the context. The snowflake data model [8] is a normalized structure for hierarchy representations. In this case, the classification scheme items are grouped by levels or entities. There are as many groups as levels the classification scheme has. Snowflake models are widely used on data warehouses to build hierarchical classifications on structures known as dimensions. Some examples of dimension are Time, Product Category, Geography, Occupations, etc. An example of snowflake data model can be seen in Figure. 4. In this pattern the example is an occupation hierarchical

[10] `http://www.w3.org/2001/sw/rdb2rdf/implementation-report/`

Professione0

ID0	Desc0
01	Professioni specialistiche e tecniche
02	Professioni operative della gestione dimpresa

Professione1

ID1	Desc1	ID0
01.05	Specialist e tecnici delle scienze informatiche	01
02.05	Specialist e tecnici delle gestione dimpresa	02

Professione2

ID2	Desc2	ID1
01.05.01	Specialist delle scienze informatiche	01.05
01.05.02	Tecnici delle scienze informatiche	01.05
02.05.01	Specialist delle gestione dimpresa	02.05
02.05.02	Tecnici delle gestione dimpresa	02.05

Fig. 4. An example of Snowflake Data Model

classification hold on three different tables, one for each level (PROFESSIONI_0, PROFESSIONI_1, PROFESSIONI_2).

The ontology generated will be based on the taxonomy architectural pattern (AP-TX-01) [11]. Each term in the classification scheme is mapped to a class, and the semantics of the relationship between children and parent terms are made explicit by using an external resource. Figure. 5 illustrates the generated ontology from the example. Note that although the transformation copies the

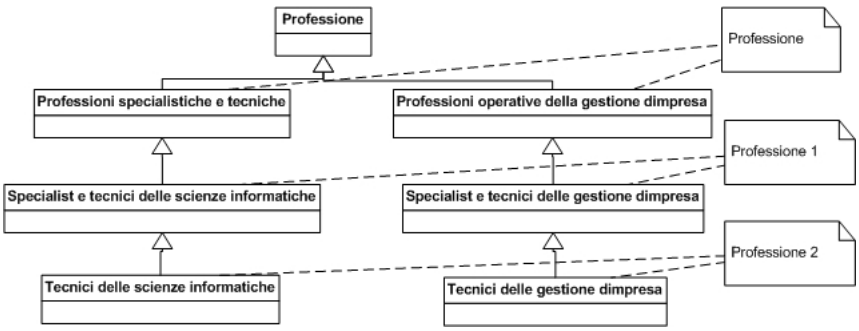


Fig. 5. Generated ontology from the example of Snowflake Data Model

the hierarchy expressed by the database content, the resulting ontologies consists only the schema (collection of classes and their labels) without their individuals. Hence, this is a T-Box transformation as we discussed in Section. 1.

Next we present the procedure that generates R2RML mapping corresponding to the pattern. First, create an rr:TriplesMap instance for every table with the table name as its rr:logicalTable value. Then create an rr:SubjectMap instance for the TriplesMap with rdfs:Class as its rr:class and concatenation of namespace base and primary key of the table as its rr:template value. Additionaly, the user may provide an rr:PredicateObjectMap instance that specifies the name of the class. If the table doesn't have a foreign key, it means that the table is mapped into the root class of the ontology. Otherwise, create an rr:PredicateObjectMap instance with rr:objectMap that joins the foreign key with the referenced primary key.

Input: The tables of database *tables*, the URI of the target class *classURI*
Output: R2RML Mapping Document MD

1: $MD \leftarrow$ createMappingDocument()
2: **for** $t \in tables$ **do**
3: $TM \leftarrow$ createTriplesMap()
4: $TM.logicalTable.tableName \leftarrow$ t.tableName
5: $TM.subjectMap.class \leftarrow$ rdfs:Class
6: $TM.subjectMap.template \leftarrow$ CONCAT(classURI, table.PKColumn)
7: $TM.predicateObjectMap[0].predicate \leftarrow$ rdfs:label
8: $TM.predicateObjectMap[0].objectMap.column \leftarrow$ t.LabelColumn
9: **if** $t.PK = \{\}$ **then**
10: $TM.predicateObjectMap[1].predicate \leftarrow$ rdfs:subClassOf
11: $TM.predicateObjectMap[1].objectMap.constant \leftarrow$ classURI
12: **else**
13: $TM.predicateObjectMap[1].predicate \leftarrow$ rdfs:subClassOf
14: $TM.predicateObjectMap[1].objectMap.constant \leftarrow$ classURI
15: **end if**
16: PUT(MD, TM)
17: **end for**
18: **return** MD

Listing 1.1. R2RML mappings corresponding to the example of Snowflake Data Model

```
<TriplesMapProfessione0> a rr:TriplesMap;
    rr:logicalTable [ rr:tableName "Professione0"];
    rr:subjectMap [ rr:class rdfs:Class; rr:termType rr:IRI;
        rr:template
            "http://example.org/resource/Professione{id0}"; ];
    rr:predicateObjectMap [rr:predicate rdfs:label;
        rr:objectMap [rr:column "desc0"];];
    rr:predicateObjectMap [ rr:predicate rdfs:subClassOf;
        rr:objectMap [ rr:constant
            "http://example.org/resource/Professione"]; ];.

<TriplesMapProfessione1> a rr:TriplesMap;
    rr:logicalTable [ rr:tableName "Professione1"];
    rr:subjectMap [ rr:class rdfs:Class; rr:termType rr:IRI;
        rr:template
            "http://example.org/resource/Professione{id1}"; ];
    rr:predicateObjectMap [ rr:predicate rdfs:label;
        rr:objectMap [rr:column "desc1"]; ];
    rr:predicateObjectMap [rr:predicate rdfs:subClassOf;
        rr:objectMap [rr:termType rr:IRI;
            rr:parentTriplesMap <TriplesMapProfessione0>;
                rr:joinCondition [
                    rr:child "id0" ;rr:parent "id0" ;]; ]; ];.

<TriplesMapProfessione2> a rr:TriplesMap;
    rr:logicalTable [ rr:tableName "Professione2"];
    rr:subjectMap [rr:class rdfs:Class; rr:termType rr:IRI;
        rr:template
```

```
        "http://example.org/resource/Professione{id2}"; ];
    rr:predicateObjectMap [ rr:predicate rdfs:label;
        rr:objectMap [rr:column "desc2"]; ];
rr:predicateObjectMap [rr:predicate rdfs:subClassOf;
    rr:objectMap [rr:termType rr:IRI;
        rr:parentTriplesMap <TriplesMapProfessione1>;
            rr:joinCondition [
                rr:child "id1" ; rr:parent "id1" ;]; ]; ];.
```

PR-NOR-TSTX-02 : Pattern for re-engineering a thesaurus following the relation-based data model into an ontology schema A thesaurus represents the knowledge of a domain with a collection of terms and a limited set of relations between them. The relation-based data model [10] is a normalized structure, in which relationship types are not defined as fields in a record, but they are simply data values in a relationship record, thus new relationship types can be introduced with ease.

As an example, the AGROVOC Thesaurus is an structured and controlled vocabulary designed to cover the terminology of all subject fields in agriculture, forestry, fisheries, food and related domains. This thesaurus is available at `http://www.fao.org/agrovoc/`. See Figure. 6 for the graphical representation of the thesaurus.

agrovocterm

TermCode	Term
1328	Paddy
1474	Cereals
3354	Poaceae
5435	Oryza
6599	Rice

termlink

TermCode1	TermCode2	LinkTypeID
5435	6599	90
5435	3354	50
6599	5435	90
6599	1474	50
6599	1328	20

linktype

LinkTypeID	LinkDesc	LinkAbr
50	Broader Term	BT
90	Related Term	RT
60	Narrower Term	NT
20	Used For	UF

Fig. 6. An example of Thesaurus Relational-based Data Model

The ontology generated will be based on the lightweight ontology architectural pattern (AP-LW-01)[11]. Each thesaurus term is mapped to a class. For the disambiguation of the semantics of the BT, NT, RT and UF relations among thesaurus terms the pattern relies on an external resource. In our case, the semantics of the BT, NT, RT and UF relations are encoded in rr:sqlQuery of the R2RML mappings. Figure. 7 illustrates the generated ontology from the example.

Next, we present the procedure to generate R2RML mappings corresponding to this pattern. First, create an rr:TriplesMap instance whose rr:logicalTable is a view of join result of Terminology table, Terms Relationship table, and Relationship Type table. We let the user decide whether the join type is INNER or LEFT OUTER. In the view, specify also the translation from Relationship Type table values into skos properties using SQL CASE. Then, create an instance of

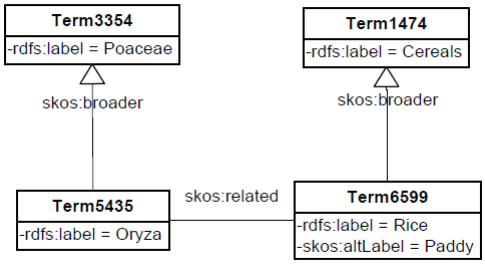


Fig. 7. The generated ontology from the example of Thesaurus Relational-based Data Model

rr:SubjectMap with rdfs:Class as its rr:class. The URI values is specified through rr:template as the concatenation of Term class namespace with the value of the table's primary key. Additionaly, an rr:PredicateObjectMap instance can be provided to specify the class name. Then, create another rr:PredicateObjectMap instance that maps a property to the the relationship between one term with another specified in the SQL CASE value provided in the view.

Input: Table Term $TblTerm$, Table Relationship $TblRel$, Table Relationship type $TblRelType$, Term Class URI $classURI$
Output: R2RML Mapping Document MD

1: $MD \leftarrow$ createMappingDocument()
2: $TM \leftarrow$ createTriplesMap()
3: $V = TblTerm \bowtie TblRel \bowtie TblRelType$
4: $TM.logicalTable.sqlQuery \leftarrow V$
5: $TM.subjectMap.class \leftarrow rdfs:Class$
6: $TM.subjectMap.template \leftarrow$ CONCAT(classURI, TblRel.TermCode1)
7: $TM.predicateObjectMap[0].predicate \leftarrow rdfs:label$
8: $TM.predicateObjectMap[0].objectMap.column \leftarrow TblTerm.Term$
9: $TM.predicateObjectMap[1].predicateMap.column \leftarrow linkURIObject$
10: $TM.predicateObjectMap[1].objectMap.template \leftarrow$
$CONCAT(classURI, TblRel.TermCode2)$
11: $TM.predicateObjectMap[2].predicateMap.column \leftarrow linkURIData$
12: $TM.predicateObjectMap[2].objectMap.template \leftarrow$
$CONCAT(classURI, TblRel.Term)$
13: PUT(MD, TM)
14: **return** MD

Listing 1.2. R2RML mappings corresponding to the example of Thesaurus Relation-based Data Model

```
<TriplesMapTerm> a rr:TriplesMap;
    rr:logicalTable [ rr:sqlQuery """
    SELECT t.TermCode, t.Term, tl.TermCode2
        , lt.LinkDesc , lt.LinkTypeID
        , CASE lt.LinkAbr
            WHEN 'BT' THEN 'skos:broader'
            WHEN 'NT' THEN 'skos:narrower'
            WHEN 'RT' THEN 'skos:related'
        END AS linkURIObject
        , CASE lt.LinkAbr
            WHEN 'UF' THEN 'skos:altLabel'
        END AS linkURIData
    FROM agrovocterm t
    LEFT OUTER JOIN termlink tl
```

```
    ON t.TermCode = tl.TermCode1
  LEFT OUTER JOIN linktype lt
    ON tl.LinkTypeID = lt.LinkTypeID
  """];

  rr:subjectMap [rr:class rdfs:Class;rr:termType rr:IRI;
    rr:template
      "http://example.org/resource/Term{TermCode}";];

  rr:predicateObjectMap [rr:predicate rdfs:label;
    rr:objectMap [rr:column "Term"];];

  rr:predicateObjectMap [rr:predicateMap [
    rr:column "linkURIObject"; rr:termType rr:IRI];
      rr:objectMap [ rr:template
      "http://example.org/resource/Term{TermCode2}"]; ];

  rr:predicateObjectMap [rr:predicateMap [
    rr:column "linkURIData"; rr:termType rr:Literal];
      rr:objectMap [ rr:template
      "http://example.org/resource/Term{Term}"]; ];
.
```

We have seen two examples of using R2RML mappings in order to generate ontologies from PR-NORs. There are other approaches for this goal, although our approach is better for several reasons. For example, using other RDB2RDF languages instead of R2RML, such as R2O [1] or D2RQ [3] can be employed. Unlike R2RML, both R2O and D2RQ do not permit the use of arbitrary SQL queries as the logical table, which can be useful for joining multiple tables which some complex conditions. Using a standard mapping language also bring benefits on the practical side, as multiple implementations are available. Even D2R system, which initially implemented as D2RQ engine, will also give support to R2RML. Other possible approach is not to use R2RML or any RDB2RDF mapping languages, but using ad-hoc approach like creating a custom program for each of the pattern. However, this approach can be considered inferior as the the reusability aspect is lower than reusing mappings, not to mention the time has to be invested to create the custom program instead of just choosing one of the available R2RML implementations, such as Morph[11].

4 Conclusions and Future Work

In this paper we have presented an approach that combines PR-NORs with R2RML mappings for building ontologies from relational database content, i.e., transforming the database content into an ontology schema by using Re-engineering Patterns and R2RML mappings. Furthermore, we have seen that besides for generating data-triples (triples that describes instances of an ontology), R2RML mappings are useful also to generate schema-triples (triples that describe the schema of an ontology).

In the future, we will continue identifying other Re-engineering patterns that may take benefit from R2RML mappings. We will explore the possibility to combine R2RML mappings with other data-source type, such as XML, CSV or

[11] https://github.com/jpcik/morph

spreadsheets. Because R2RML mappings are RDF documents, it is possible to store those mappings in a triple store. Once those mappings have been stored in the triple store and annotated with meta-data properties, users can pose query the triple store in order to get all mappings correspond to a specific pattern. For example, user can pose a SPARQL query `SELECT * FROM {?m a mapping. m hasPattern ?p. FILTER REGEX(?p, "Snowflake")}`, and the answers will be all mappings corresponding to the snowflake data model pattern. The system is currently under development and being populated with mappings and patterns and we plan to report this system as future work.

Acknowledgments: This work has been supported by the PlanetData (FP7-257641), BabelData (TIN2010-17550), and myBigData (TIN2010-17060) projects. We would like to kindly thank all W3C RDB2RDF Working Group members.

References

1. J. Barrasa, Ó. Corcho, and A. Gómez-pérez. R2o, an extensible and semantically based database-to-ontology mapping language. In *in In Proceedings of the 2nd Workshop on Semantic Web and Databases (SWDB2004*, 2004.
2. D. Beckett and T. Berners-Lee. Turtle-terse rdf triple language. *Syntax*, 28(January), 2008.
3. C. Bizer and A. Seaborne. D2rq-treating non-rdf databases as virtual rdf graphs. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, page 26, 2004.
4. C. Caracciolo, J. Heguiabehere, V. Presutti, and A. Gangemi. Initial network of fisheries ontologies. Technical report, NeOn project deliverable D7.2.3, 2009.
5. C. Date. *An introduction to database systems*, volume 2. Addison Wesley Publishing Company, 1983.
6. B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep web. *Commun. ACM*, 50(5):94–101, May 2007.
7. M. Hert, G. Reif, and H. Gall. A comparison of rdb-to-rdf mapping languages. In *Proceedings of the 7th International Conference on Semantic Systems*, pages 25–32. ACM, 2011.
8. E. Malinowski and E. Zim·nyi. Hierarchies in a multidimensional model: From conceptual modeling to logical representation. *Data and Knowledge Engineering*, 2006.
9. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau Jr, S. Auer, J. Sequeda, and A. Ezzat. A survey of current approaches for mapping of relational databases to rdf. *W3C RDB2RDF XG Incubator Report*, page W3C, 2009.
10. D. Soergel. Data models for an integrated thesaurus database. *Comatibility and Integration of Order Systems*, 24(3):47–57, 1995.
11. M. C. Suárez-Figueroa, S. Brockmans, A. Gangemi, A. Gómez-Pérez, J. Lehmann, H. Lewen, V. Presutti, and M. Sabou. Neon modelling components. Technical report, NeOn project deliverable D5.1.1, 2007.
12. B. Villazón-Terrazas. *A Method for Reusing and Re-engineering Non-ontological Resources for Building Ontologies*. IOS Press, 2012.

The Template Instance Pattern

Csongor Nyulas, Tania Tudorache, Samson Tu

Stanford Center for Biomedical Informatics Research, Stanford University, US
{nyulas, tudorache, swt}@stanford.edu

Abstract. We present the Template Instance Pattern, a content design pattern that marks an instance, which is used as a value for a property, to be a "template". A template instance is intended to be immutable (none of its properties can be changed). If the content (i.e., any of the property values) of the template instance needs to be changed, a template flag will indicate to an ontology editor that it should create a clone of the template instance, and replace the value of the property with the desired value in the newly created clone. This pattern is especially useful for cases in which an ontology makes abundant use of reified relations (represented as instances), which are repetitive and that increase significantly the size of an ontology. We created this pattern as a result of building a large real world medical ontology that makes excessive use of reification.

1 Background

Ontologies make use of reified relations to model n-ary relations or describe additional properties of a relation (e.g., confidence level, provenance, and so on). For example, biomedical ontologies often need to qualify facts about the domain with scientific evidence: a definition of a disease would need to include links to scientific papers that sourced or endorsed that definition, or relationships between different entities need to carry a probability. Some ontologies, such as the ICD-11 [2, 1] make extensive use of reified relations. Practically, all relations in the ICD-11 ontology are reified. One issue with reification is that it creates the intermediate reified instances that do not contribute any real content, but they rather provide a structure that groups together all properties of a reified relation. An ontology that makes abundant use of reified relationships is likely to be very large in size due to the "clutter" introduced by the reified instances. A larger ontology may be harder to maintain, and may create challenges for ontology editors or reasoners. The template instance pattern proposes a way of reducing the number of reified instances and the related property assertion axioms in an ontology, especially for the cases in which the reified relations are identical for multiple entities.

2 Solution Description

The pattern proposes to use the same "template" instance as the value of a reified property for multiple subjects (rather than having multiple copies of the same reified instance). We give an example in Figure 1 to make it easier to understand. The individuals `x1`, `x2` and `x3` have a reified property `p` that has as values the reified individuals, `y1`, `y2` and `y3`, respectively. In OWL, we would have the following object property assertions:

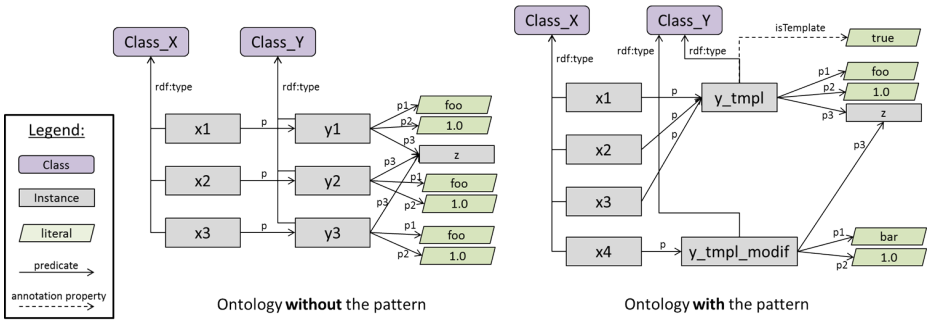


Fig. 1. An abstract example of reified modeling without (left side) and with (right side) the Template Instance Pattern.

```
(x1 p y1)
(x2 p y2)
(x3 p y3)
```

The properties for the reified individuals, `y1`, `y2` and `y3` are all identical (the underscore represents the index for `y`: 1, 2, or 3):

```
(y_ p1 foo)
(y_ p2 1.0)
(y_ p3 z)
```

The pattern proposes to create a template instance, `y_tmpl` that has the common property values (right hand side of Figure 1):

```
(y_tmpl p1 foo)
(y_tmpl p2 1.0)
(y_tmpl p3 z)
```

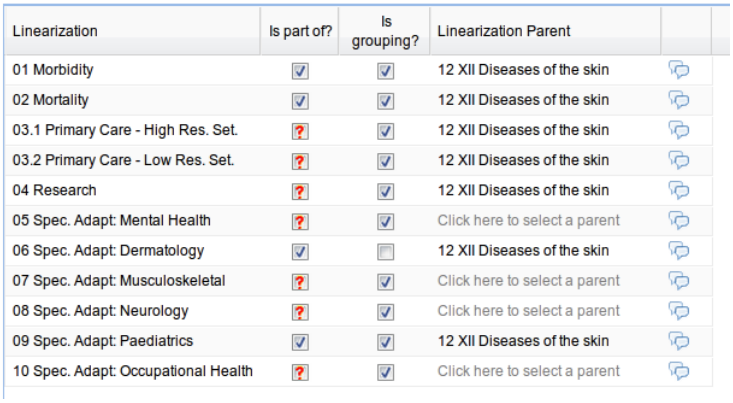
The template instance would be used as the value for the reified property for `x1`, `x2` and `x3`:

```
(x1 p y_tmpl)
(x2 p y_tmpl)
(x3 p y_tmpl)
```

In addition, we will also add one annotation property, `isTemplate:true`, on `y_tmpl` to mark that it as a template instance. The intention is that the template instance is immutable, i.e., the property values of the template instance cannot be changed.

In the case that a user would like to change a property value of the reified instance, a clone of the template instance would be created and the change would occur on the clone. An ontology editor would use the `isTemplate` annotation property to check that a certain instance cannot be modified, and it should rather create a clone.

Figure 1 gives an example of this situation. Say that at a given time t_0, there was an additional `x4` individual that had the template instance as the value for `p`. At t_0:



Linearization	Is part of?	Is grouping?	Linearization Parent
01 Morbidity	☑	☑	12 XII Diseases of the skin
02 Mortality	☑	☑	12 XII Diseases of the skin
03.1 Primary Care - High Res. Set.	?	☑	12 XII Diseases of the skin
03.2 Primary Care - Low Res. Set.	?	☑	12 XII Diseases of the skin
04 Research	?	☑	12 XII Diseases of the skin
05 Spec. Adapt: Mental Health	?	☑	Click here to select a parent
06 Spec. Adapt: Dermatology	☑	☐	12 XII Diseases of the skin
07 Spec. Adapt: Musculoskeletal	?	☑	Click here to select a parent
08 Spec. Adapt: Neurology	?	☑	Click here to select a parent
09 Spec. Adapt: Paediatrics	☑	☑	12 XII Diseases of the skin
10 Spec. Adapt: Occupational Health	?	☑	Click here to select a parent

Fig. 2. Proposed usage of the Template Instance Pattern for representing views in the ICD-11 ontology. Each row in the table represents a reified instance (which represents a view), and each column represents a property value of the reified instance. Each class in the ontology can be part of one or several views.

```
(x4 p y_tmpl)
(y_tmpl p1 foo)
(y_tmpl p2 1.0)
(y_tmpl p3 z)
```

Later, at time t_1, the user decides to change the value `p1` from `foo` to `bar`. The ontology editor (or the user, if there is no support in the tool), will see the `isTemplate` annotation property on `y_tmpl` as set on `true`, and it will create a clone of it, `y_tmpl_modif`, by cloning also all the object and data property axioms. Then, it will change the value of `p1` on `y_tmpl_modif`. At t_1:

```
(x4 p y_tmpl_modif)
(y_tmpl_modif p1 bar)
(y_tmpl_modif p2 1.0)
(y_tmpl_modif p3 z)
```

As a result of using the pattern, we can reduce the number of data and object property axioms in the ontology. This "deflation" of the ontology is especially significant, if there are many repetitive values in the ontology, with few changes, and if the number of properties of the reified instance is large.

3 Example of Usage

As we have mentioned before, the ICD-11 ontology that describes diseases and their properties makes extensive use of reification. A class in the ICD-11 ontology represents a disease, which has several properties (e.g., title, definition, synonyms, signs and symptoms, etiology, manifestation, etc.) that are encoded as property axioms. Each disease class can participate in one or several "views" that will be extracted from the ICD-11

ontology. A view (in ICD language called "linearization") is a portion of the ontology that is relevant for a particular sub-domain or use case (e.g. Morbidity, Mortality, Primary Care, Dermatology, etc.). Figure 2 shows an example of these views. For each disease class, the specifications of the views it participates in, are represented as reified instances (the table rows in Figure 2). Each of the view specification instances contains five additional properties (partially shown as columns in the table). Each disease class has to specify their inclusion in ten different views. Each view is represented as a reified instance. The ontology currently contains over 40.000 disease classes, each having 10 view specifications, and each of the view specifications containing additional 5 properties. Currently, we have over 400.000 reified view specifications, and over 2 million property assertion describing them. Many of these reified view specification instances are identical. We plan to use the Template Instance Pattern to reduce significantly the number of the reified instances in the ontology. For example, for the initial modeling (that contains no user modifications), we would have only 10 reified view specification instances (reduced from 400.000) and 400.000 property assertions (reduced from 2 million). This "deflation" of the ontology size is significant, would allow us to more easily maintain the view specifications, and would have a beneficial impact on the performance of the ontology tools used for editing ICD-11.

4 Conclusion

We presented the template instance pattern that was created from the need to address scalability and maintainability issues in the development of the ICD-11 ontology. The pattern proposes the use of template instances as values for reified properties, and their flagging using an annotation property that can be used to provide support for this pattern in ontology tools. We also described an example of the pattern and its operationalization in a generic ontology tool.

References

1. T. Tudorache, S. Falconer, C. Nyulas, N. Noy, and M. Musen. Will Semantic Web Technologies Work for the Development of ICD-11? In *The 9th Intl. Semantic Web Conference (ISWC 2010)*, pages 257–272. Springer, 2010.
2. World Health Organization. The 11th Revision of the International Classification of Diseases (ICD-11). `http://www.who.int/classifications/icd/revision/en/index.html`. Last accessed: August, 2012.

Conformance to standards

Monika Solanki * and Craig Chapman
Knowledge Based Engineering Lab
Birmingham City University, UK
{monika.solanki, craig.chapman}@bcu.ac.uk

Abstract. In this paper we present the **S**tandards **E**nforcer **P**attern (SEP). The remit of SEP is to enable the ontological modelling of processes, activities, operations and services that enforce guideline(s) recommended by a specific standard and need to explicitly indicate their conformance to it. The pattern allows the inclusion of minimalistic information regarding the conformance, while retaining the flexibility to extend the ontological primitives as required. As an exemplifier for the pattern, we present a use case from the algal biomass domain. We model the process of algal biomass production that enforces the Minimum Descriptive Language (MDL) standard for algal operations.

1 Introduction

Activities, operations, processes and services in most domains of interest are governed by standards. The objective of a standard is to ensure consistency in implementations and uniformity in quality by ensuring the **repeated** and continuous use of prescribed rules and guidelines. The ISO/IEC Guide 2:1996 [1], definition 3.2 defines a standard as a set of specification that is "established by consensus and approved by a recognized body that provides for common and repeated use, rules, guidelines or characteristics for activities or their results, aimed at the achievement of the optimum degree of order in a given context".

In order to provide a generic mechanism for the inclusion of ontological modelling primitives of conformance to standards, independent of the domain of application and the context of processes, we propose the content ontology design pattern **S**tandards **E**nforcer **P**attern (SEP).

2 Standards Enforcer Pattern (SEP)

2.1 Intent

The remit of the SEP content pattern is to represent the relation between standards and the processes, operations, activities and services that enforce them,

* Principal and corresponding author
[1] http://www.etsi.org/WebSite/Standards/WhatIsAStandard.aspx

the domains they cater to and the scope of that specific process, operation, activity, service within the context of the domain.

2.2 Competency Questions:

- Which are the standards enforced by this process?
- Which processes enforce these standards ?
- What is/are the domain level scope(s) of the standard?
- Within the context of the domain what is the scope of the process, activity, operation and service to which the standard is applicable?
- What are the prescribed guidelines for a standard?
- Which prescribed guideline(s) of a standard does a specific process conform to?

2.3 Some Conceptual Elements

- `Standard`: A specification established through domain expert consensus that prescribes a set of rules and guidelines for a given contextual activity within a domain. The standard must be described informally or formally in a written document.
- `Guideline`: An entity defining a guideline included in a standard. Guidelines are usually prescribed as clauses in the written document for the standard.
- `StandardEnforcingProcess/Operation/Activity`: The domain specific entity which enforces one or more guidelines from one or more standard.
- `DomainScope`: The domain/industry/paradigm for which the standard has been designed.
- `ProcessScope`: The activity within a specific domain/industry/paradigm which is governed by the process, e.g., algae harvesting activity which is part of the biomass production process in the domain of biofuels, shielded metal arc welding used in the production of tools in the manufacturing domain.
- `enforcesStandard`: The relationship between the enforcing process and the standard.
- `enforcedBy`: The relationship between the standard and the enforcing process. This is an inverse relationship to `enforcesStandard`.
- `hasDomainScope`: The relationship linking the standard with the domain to which it is applicable. A standard can cover multiple domains.
- `hasProcessScope`: The relationship linking the standard enforcing process with the scope of the process. A standard enforcing process can include multiple process scopes.
- `hasDescriptionDocument`: The relationship linking the standard to the real world document that informally/formally describes it. The object value for this property is a pointer (URI) to the document resource representing the standard.
- `hasDescriptionClause`: The relationship linking a guideline to the prescribing clause in the real world document for the standard. The object value for this property is a pointer (URI) to the clause in the document resource representing the standard.

2.4 Pattern Representation

The Manchester syntax rendering for the concept `Standard` are illustrated below:

```
Class: Standard
    EquivalentTo:
        (hasDomainScope some DomainScope)
         and (prescribesGuideline some Guideline)
         and (hasDescriptionDocument min 1 owl:Thing)
    SubClassOf:
        isEnforcedBy min 0 ProcessEnforcingStandar
```

Note that in the definition of a standard, we require that it includes the scope of the domain, guidelines and the description document that informally defines the standard.

Figure 1 illustrates the graphical representation of SEP [2] [3].

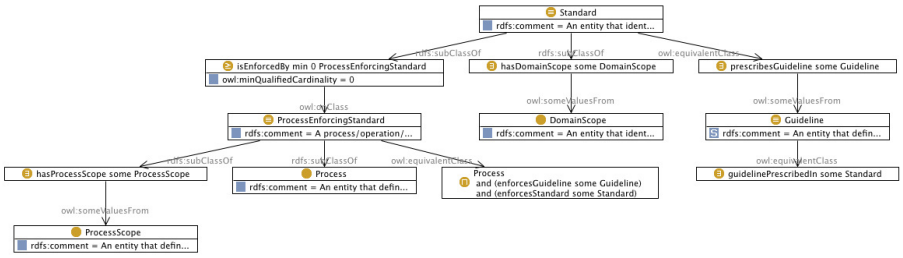


Fig. 1. Graphical Representation of **S**tandards **E**nforcer **P**attern (SEP)

2.5 Consequences

The pattern can be applied to use cases in all those domains where a standard is enforced to regulate processes. The main advantage of this pattern is that it provides the capability to link processes, operations, activities and services to

[2] The OWL ontology for the pattern is available at http://purl.org/biomass/SEP

[3] Graphical representations of the pattern in this paper have been produced using a trial version of the Maestro edition of TopBraid Composer.

their governing standards in a generic and "compositional" manner. In some scenarios it is possible that a process or an operation does not enforce all prescribed guidelines but enforces atleast some. The pattern accounts for that through the definition of the process enforcing the standard.

2.6 Example usage: Algal Biomass Domain

As an exemplifier for SEP, we present a use case from the domain of algal biomass production. Figure 2 depicts the application of SEP to an ontology that models algal biomass production. The "Minimum Descriptive Language"(MDL) standard[4] proposed by the Algal Biomass Association is enforced by the production operation. MDL recommends a set of descriptive metrics to uniformly characterise the analysis of large scale algal operations. In this use case, the ontology defines the concepts and relationships for the operation and incorporates SEP by enforcing a guideline for measuring Carbon input to the operation.

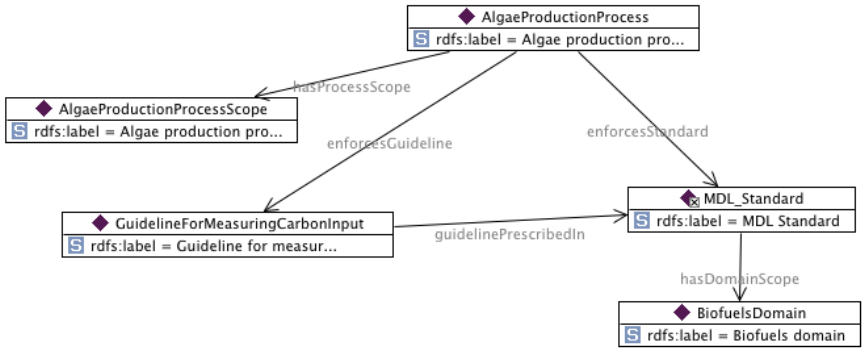


Fig. 2. Graphical Representation of SEP as applied to the domain of algal biomass production

3 Summary

SEP provides a mechanism to ontologically declare the conformance of a process with one or more standards. The pattern is flexible and compositional. It can be exploited to include few or more guidelines from multiple standards and can be easily combined with other patterns.

[4] `http://www.algalbiomass.org/`

Reactive Processes

Monika Solanki* and Craig Chapman
Knowledge Based Engineering Lab
Birmingham City University, UK
{monika.solanki, craig.chapman}@bcu.ac.uk

Abstract. In this paper we present the *Reactor Pattern* to enable the modelling of processes that consume inputs and produce outputs under specific environmental conditions and on being triggered by certain events. Reactor pattern is a content ontology design pattern and is especially targeted towards modelling reactive processes with a "black box" view of the process.

1 Introduction

Many scenarios in the engineering, manufacturing and biotechnologies sectors employ "reactive" processes, usually carried out in a closed system, e.g., a bioreactor in which a chemical process is carried out, which involves organisms or biochemically active substances derived from such organisms. Such processes consume inputs and produce outputs in a controlled environment and on being triggered by certain events. The purpose of the reactor pattern is to enable the ontological modelling of such reactive processes in a generic way across multiple domains. The reactor pattern is a content design pattern and provides ontological placeholders for input and output parameters, environmental conditions and events. The pattern exploits other CPs for the definition of certain entities.

2 Reactor Pattern

2.1 Intent

The remit of the reactor pattern is to enable the modelling of processes that are reactive, consume inputs and produce outputs under specific environmental conditions a.k.a. constraints and on being triggered by certain events. Reactive processes are parametric where the governing parameters are process inputs and outputs. The pattern can be instantiated to provide a knowledge level solution to the problem of capturing parametric process related information in a domain independent way.

* Principal and corresponding author

2.2 Competency Questions:

- What are the "types" of inputs consumed by a certain process?
- What are the "types" of outputs produced by a certain process?
- What are the values of parameters for a certain process?
- What is the measurement criteria for a specific parameter?
- What environmental conditions need to hold for the process to get activated?
- Which event triggers a specific process?

2.3 Some Conceptual Elements

- `Process`: placeholder for a process. The concept covers the definition of a generic process.
- `ParametricProcess`: placeholder for a process governed by parameters.
- `ReactiveProcess`: a process specialising from `ParametricProcess`. Note that in our abstraction, a reactive process needs to explicitly define at least one input and output.
- `ProcessParameter`: an overarching entity representing parameters consumed and produced by the process. The concept extends from `Parameter` defined in the Parameter[1] CP.
- `InputParameter`: a specialisation of the `ProcessParameter` representing the input parameter. A process can consume several inputs.
- `OutputParameter`: a specialisation of the `ProcessParameter` representing the output parameter. A process can produce several outputs.
- `EmvironmentalCondition`: an entity representing environmental conditions governing the activation of the process. The condition may be specified as a SWRL rule[2]. There can be several environmental conditions for a process.
- `Event`: an event that triggers the process.
- `hasEnvironmentalCondition`: a relation between the environmental condition and the process.
- `triggeredBy`: a relation between the process and the environment.
- `definesCondition`: a relation between, `EnvironmentalCondition` and one or more conditions it enforces.

It is worth noting that we explicitly abstract from providing further details on how the environmental conditions are represented or how the parameter measurements are defined. These are not part of the pattern definition. Well defined existing vocabularies such as the LODE ontology[3] for modelling events and the QUDT vocabulary[4] for measurement units should be exploited to provide definitions for these concepts.

[1] `http://www.ontologydesignpatterns.org/cp/owl/parameter.owl`
[2] http://www.w3.org/Submission/SWRL/
[3] http://linkedevents.org/ontology/
[4] http://qudt.org/1.1/vocab/dimensionalunit

2.4 Pattern Representation

The core concept in the pattern is a reactive process, parameterised with inputs and outputs. A Manchester syntax rendering of the concept is illustrated below.

```
Class: ReactiveProcess
    EquivalentTo:
        ParametericProcess
         and (hasInputParameter some InputParameter)
         and (hasOutputParameter some OutputParameter)
         and (triggeredBy some Event)
   SubClassOf:
        hasEnvironemntalCondition min 0 EnvironmentalCondition,
        ParametericProcess
```

Figure 1 depicts the graphical representation of the reactor pattern [5] [6].

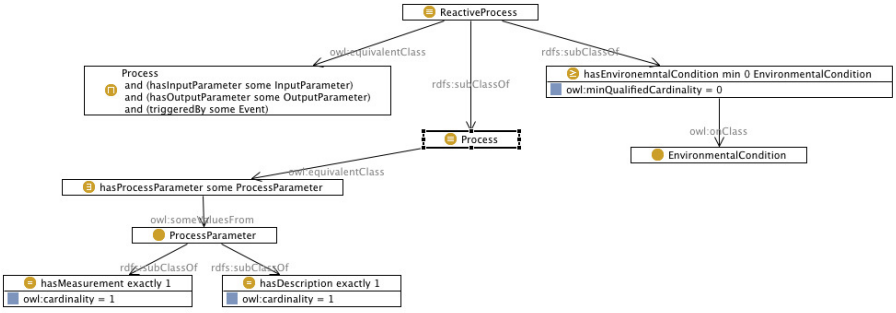


Fig. 1. Graphical Representation of Reactor Pattern

2.5 Consequences

The main advantage of this pattern is that its provides ontological modelling capabilities for the inputs, outputs and environmental conditions that govern reactive processes across several domains, independent of modelling details of the actual reactor involved. This effectively caters for exposing a black box view of the process, which is very desirable when querying the model for consumption and production logistics of the process.

[5] The OWL ontology for the pattern is available at http://purl.org/biomass/ReactorPattern

[6] Graphical representations of the pattern in this paper have been produced using a trial version of the Maestro edition of TopBraid Composer.

2.6 Example usage: Algal Biomass Domain

As an exemplifier for the reactor pattern, we present a use case from the domain of algal biomass. The set of inputs to the process of algal biomass cultivation are carbon, water, total infrastructure area, total energy, nutrients, consumables and labour. Possible outputs from the process are algal constituent products, indirect algal products, uncaptured gas emission, liquid waste output, solid waste output. Some environmental conditions that must hold for the algae to be harvested are,

- The water must be in a temperature range that will support the specific algal species being grown.
- The pH range for most cultured algal species should be between 7 and 9, with the optimum range being 8.2-8.7.

The event that triggers of the algae cultivation is the addition of the source culture to the growing containers or reactors. Figure 2 depicts the application of the reactor pattern.

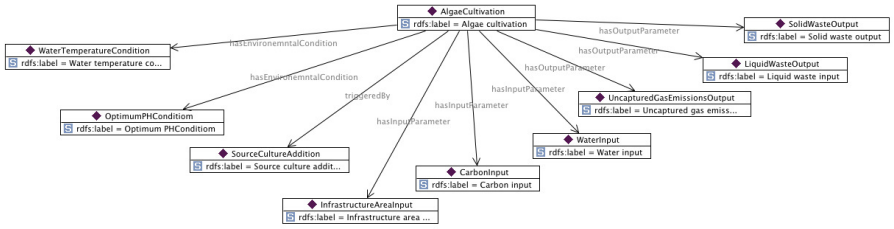


Fig. 2. Graphical Representation of reactor pattern for modelling the algal cultivation process

3 Summary

The reactor pattern provides a building block for the ontological modelling of reactive processes. The pattern can be used across domains in scenarios where a reactor is used to run processes that consume inputs to produce outputs under controlled environmental conditions and when triggered by certain events. As an example, the pattern has been applied to the algal biomass domain to model the reactive process of algae cultivation.