Open Access

Open Journal of Semantic Web (OJSW)
Volume 2, Issue 1, 2015

http://www.ronpub.com/ojsw
ISSN 2199-336X

Ontology Evolution Using Ontology Templates

Miroslav Blaško, Petr Křemen, Zdeněk Kouba

Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Technicka 2, 16000 Prague, Czech Republic, {miroslav.blasko, petr.kremen, kouba}@fel.cvut.cz

Abstract

Evolving ontologies by domain experts is difficult and typically cannot be performed without the assistance of an ontology engineer. This process takes long time and often recurrent modeling errors have to be resolved. This paper proposes a technique for creating controlled ontology evolution scenarios that ensure consistency of the possible ontology evolution and give guarrantees to the domain expert that his/her updates do not cause inconsistency. We introduce ontology templates that formalize the notion of controlled evolution and define ontology template consistency checking service together with a consistency checking algorithm. We prove correctness and demonstate the practical use of the techniques in two scenarios.

Type of Paper and Keywords

Regular research paper: *ontology template, ontology evolution, reasoning, consistency checking, semantic web*

1 Introduction

With the growing amount of data, formal ontologies are getting more and more popular for capturing shared meaning of heterogeneous and incomplete data and subsequent data integration. These features distinguish open-world ontologies from closed-world relational databases. Going inside, ontologies provide more expressive constructs to capture data meaning, not only on instance level (particular cases) but also on schema level (general knowledge). Logic-based formalization of ontologies is exploited during automated reasoning, in particular for inferring new knowledge, powerful consistency checking [2], query answering [34], or error explanation [21].

Ontology engineering (and ontology design in particular) is a complex engineering discipline, requiring deep knowledge of the particular domain as well as familiarity with top-level ontologies, existing domain ontologies and design methodologies. During ontology design domain experts and computer scientists cooperate. As a result of some ontology engineering methodologies like Methontology [13], a *core schema* is designed in cooperation of both parties, possibly with use of *ontology design patterns* [15]. However, an ontology is a dynamic body and evolves in time as new knowledge comes. The evolution process is tackled in existing ontology engineering methodologies on the high level, requiring among others ontology consistency to be maintained.

In order to make ontology evolution efficient domain experts can be given tools to enrich the ontology themselves. For example, in the MONDIS project [7, 5] we came across a scenario where domain experts needed to develop taxonomical knowledge. Although they were able to perform this task (e.g. they developed the taxonomy of building components and materials), they weren't able to judge the computational impact of the evolution. Due to unknown nature of the evolution, ontology consistency has to be checked on the fly which slows down the work. Even worse, in case ontology enrichment results in an inconsistency/unsatisfiability or any other undesired inference, it has to be debugged and explained by standard techniques [21]. This process is typically

hard to understand and interpret by domain experts and requires assistance of computer scientists during all ontology evolution scenarios.

To cope with these problems, we introduce an idea of *controlled ontology enrichment* and formalize it into the notion of *ontology templates* – compact representation of controlled evolution scenarios. Ontology templates define simple ontology evolution scenarios for which the domain expert can be sure that the knowledge he/she creates doesn't cause inconsistency. Possible impact of the ontology templates on the ontology consistency is precomputed in advance, before the domain expert starts evolving the ontology.

We will explain motivation as well as formalization of our framework on real-case example of MONDIS project. Within the project, a core ontology [3] for description of monuments were created and later enriched with MONDIS-specific taxonomies. The domain experts were asked to add general knowledge about the developed taxonomies using specific relations of the core ontology. An example of such relation is "*hasMaterial*" relation expressing that a building component (object described by concept "*Component*") is made of a building material (object described by concept "*Material*"). Using the relation it can be stated at terminological level, which building component types may/may not be associated with which building material types. For example, it can be said that an object of type pillar may be made of a wood, but may not be made of a glass. Within the paper, we will illustrate on this running example how this specific evolution scenario can be formalized by *ontology templates* and how it can be used to check its consistency in advance – thus possibly providing human-readable explanations or fixes of the modeling errors.

Section 2 refers to the work related to our approach. In section 3, necessary background on formal ontology languages and error explanation is introduced in order to understand section 4 that presents the formal model of ontology evolution. In section 5, reasoning services are shown and ontology template validation algorithms are presented. Section 6 shortly presents the prototypical implementation of the ideas and section 7 demonstrates the techniques on two cases. The paper is concluded by section 8.

2 Related Work

This section provides an overview of related work in the area of template-based management and second-order reasoning over ontologies.

Ontology Pre-Processor Language (OPPL) [12] is a domain-specific macro language for manipulation of ontologies written in OWL. There exists an Java API to use the language within an application as well as plugins for a major OWL editor Protege [18]. The plugins allow for defining macros – operations (such as add/remove axiom, rename entity) on an ontology, which are parametrized by variables. Variable values can be assigned to the macros by OPPL query, or manually by setting the values in a form.

The OPPL Patterns plugin [20] provides mechanism to find out whether the macro-based modification of an ontology is safe – it cannot change interpretation of existing symbols within the ontology. The analysis outputs which variables can take symbols from the signature of the ontology and which variables must be outside the signature in order for the modification to be safe. This provides certain insight into possible effect of the macro to an ontology engineer before applying it. Another approach [10] using OPPL defines common anti-patterns that can be used as additional information to ontology debugging services. When a modeling error occurs anti-patterns can help explaining the error on a higher level. If there are more explanations of modeling errors, the explanations are prioritized such that anti-pattern based explanations are used first.

A major RDF-based ontology editor TopBraid Composer integrates SPIN [25] templates to parametrize operations over ontologies. SPIN templates are parametrized SPARQL queries that can be serialized into RDF. They are useful to attach additional validation constraints or inference rules to OWL classes, in order to validate/infer over their instances. However, validation constraints at the axiom level are by design not very convenient to use. The approach [8] uses Description Logics with Temporal Logic operators to formally characterize and reason about ontology evolution. The approach allows for expressing some statements about previous snapshots of an ontology (i.e. whether a concept was satisfiable in any snapshot or all previous snapshots).

There are many approaches that extends Description Logic [11, 29, 27] in order to provide reasoning with higher-level constructs. However, reasoning is incomplete for second-order logic, thus they provide either too weak extension [14] towards second-order or they extend too simple description logic (such as [1]). Hence such approaches are not applicable to our use case. A paper [9] provides a second-order framework and related calculus to unify and solve many non-standard reasoning tasks in Description Logics such as concept unification, concept contraction etc.

3 Background

This section provides an overview of ontological background needed to understand the following sections.

An ontology, within scope of this paper, is a specification of conceptualization [16] – an abstract and simplified view of the world shared between applications or people. It uses declarative logic-based formalism to describe entities and relationships between those entities within the domain of interest. Statements about the domain are called axioms.

3.1 $\mathcal{ALC}$ DL - Syntax and Semantics

Current ontology standards, like OWL 2 [30], are based on description logics. Description logics are monotonic and decidable subsets of first-order predicate logic typically aimed at knowledge representation. Although our framework is not dependent on any particular variant of Description Logics ($\mathcal{DL}$), we will use the $\mathcal{ALC}$ language [32], the basic description logic capturing fundamentals of more expressive formalisms used in the semantic web domain.

An ontology is represented in terms of *individuals* (representing *particulars*, i.e. concrete objects), *concepts* (representing *universals*, i.e set of objects) and *roles* (representing relations between objects).

$\mathcal{ALC}$ Syntax: The vocabulary of $\mathcal{ALC}$ consists of three disjoint sets of *concept names* (N_C), *role names* (N_R), and *individual names* (N_I). Let A be a concept name and R a role name. Any $\mathcal{ALC}$ concept can be constructed inductively by the following syntax rules:

$$C \rightarrow A \mid \top \mid \bot \mid \neg C \mid \exists R.C \mid \forall R.C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \tag{1}$$

An ontology consists of finite set of axioms – *concept inclusion axioms* of the form $C_1 \sqsubseteq C_2$, *class assertions* of the form $C(a)$, and *role assertion* of the form $R(a, b)$.

$\mathcal{ALC}$ Semantics: Formal semantics of $\mathcal{ALC}$ is defined by *interpretation* $\mathcal{I}$ which is a pair $\langle \Delta^{\mathcal{I}}, .^{\mathcal{I}} \rangle$. A non-empty set $\Delta^{\mathcal{I}}$ denotes domain of interpretation. $.^{\mathcal{I}}$ is an interpretation function that maps each concept name $A \in N_C$ into a subset of $\Delta^{\mathcal{I}}$, and each role name $R \in N_R$ into a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended for concepts in the following way :

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C_1 \sqcap C_2)^{\mathcal{I}} &= (C_1)^{\mathcal{I}} \cap (C_2)^{\mathcal{I}} \\
(C_1 \sqcup C_2)^{\mathcal{I}} &= (C_1)^{\mathcal{I}} \cup (C_2)^{\mathcal{I}} \\
(\exists R.\top)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b.\langle a, b\rangle \in R^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b.\langle a, b\rangle \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}
\end{aligned}$$

We say that an ontology $\mathcal{O}$ is *consistent* if there is an interpretation $\mathcal{I}$ such that for every axioms α, $\mathcal{I}$ entails α. Formally, $\forall \alpha \in O, \mathcal{I} \vDash \alpha$ such that:

$$\begin{aligned}
\mathcal{I} \vDash (C_1 \sqsubseteq C_2) \in \mathcal{O} \quad &\text{iff} \quad (C_1)^{\mathcal{I}} \subseteq (C_2)^{\mathcal{I}} \\
\mathcal{I} \vDash C(a) \in \mathcal{O} \quad &\text{iff} \quad a^{\mathcal{I}} \in C^{\mathcal{I}} \\
\mathcal{I} \vDash R(b, c) \in \mathcal{O} \quad &\text{iff} \quad \langle b^{\mathcal{I}}, c^{\mathcal{I}} \rangle \in S^{\mathcal{I}}
\end{aligned}$$

An interpretation $\mathcal{I}$ that satisfies $\mathcal{O}$ is called *model* of $\mathcal{O}$. An ontology $\mathcal{O}$ entails an axiom α if every model of $\mathcal{O}$ satisfies the axiom α. $\mathcal{O}$ entails an ontology $\mathcal{O}$‘ if every model of $\mathcal{O}$ is also model of $\mathcal{O}$‘.

In addition to ontology consistency and entailment checking, more complicated reasoning services that make use of consistency checking are necessary in practical cases. We will need the error explanation service that can explain ontology inconsistency in terms of a *minimal inconsistency preserving set* (MIPS)[1].

Formally, for an inconsistent ontology $\mathcal{O}$, a MIPS is a set of axioms $\mu \sqsubseteq \mathcal{O}$ which is *inconsistent*, and *minimal* with this property, i.e. $\forall \alpha \in \mu : (\mu \setminus \{\alpha\})$ is consistent. Generally, an inconsistent ontology can have many "roots of inconsistency" corresponding to many MIPSes. Computing all of them is an exponential problem in the number of ontology consistency checks [23]. Fortunately, computing a single MIPS is polynomial, as shown in [23]. We will use this technique for computing a single MIPS during evolution schema consistency checking in Section 5.

In addition, we will need a technique for solving a well known NP-Complete problem, i.e. finding *minimal hitting set* [31] [24] of some $\mathcal{X}$, where $\mathcal{X}$ is a set of subsets of a finite set S. *Minimal hitting set* of $\mathcal{X}$ is a minimal set S' such that $S' \subseteq S$ and S' contains at least one element from each subset in $\mathcal{X}$.

3.2 Activities Related to Ontologies

Activities related to ontologies that are relevant to our work can be defined according to [36] as follows :

- *Ontology management* is set of methods and techniques that support creating, modifying, versioning, querying, and storing ontologies.
- *Ontology modification* refers to an activity of changing the ontology without considering its consistency of ontologies.

[1] In literature, concept unsatisfiability, or general entailment explanations are often considered [22], instead of ontology consistency explanation. We will use just the latter notion, as both concept unsatisfiability and entailments can be reduced to ontology inconsistency problems as shown in [2].

- *Ontology evolution* refers to an activity of facilitating the modification of an ontology by preserving its consistency.

4 Formal Model of Ontology Templates

As stated in introductory part of this paper, there are reasons for inclusion of additional semantics to parts of the ontology and in order to support extended reasoning, it must be done in a formal way.

First, the grammar of logic-based ontology languages is typically designed to find compromise between contradicting requirements such as support for reasoning, simplicity of grammar constructors, and ease of adoption by software tools. For example to support reasoning, ontology languages restrict the grammar to constructs that in most or all cases make reasoning decidable. Thus, basic building block of an ontology language, an ontology axiom, is usually not convenient to express some statement in person's mind. Especially in cases, where ontology contains a lot of complex and recurring structural patterns, it is easier to think of such patterns as wholes.

Second, there are many cases, where possible evolution of ontology by an application can be described in terms of atomic operations on the ontology. If we are able to formalize possible space of the evolution, we can apply reasoning techniques of the source language to provide inference or debugging services of the evolving ontology in advance.

Formal model able to describe mentioned evolution and its reasoning services must be able to describe: initial setting from which ontology evolution starts; atomic operations of the evolution using some higher level constructs ("a template"); a way to express new inference and debugging services in terms of defined atomic operations.

An ontology evolution starts from an initial ontology that is being modified by addition and removal of axioms. We assume that each atomic operation of the evolution adds/removes predefined axiom set called *axiom template grounding*, where *axiom template* defines family of such sets. Moreover, we assume that the initial ontology will be the stable part of the evolution, i.e. no axioms from this ontology will be removed during the evolution. This stable initial ontology we call a *core ontology* of the evolution.

Recall the description of ontology activities from Section 3.2. In the same sense with respect to scenario in the previous paragraph we will use terms *template-based ontology modification* – an modification using *ontology templates*, when consistency of the ontology is not considered and *template-based ontology evolution* – an modification using *ontology templates*, when the consistency of the ontology is considered. In addition, we will use term *template-based ontology extension* as both an activity and an ontology that was created from a *core ontology* by additions of some *ontology template groundings*. In the following text we define described terms formally.

4.1 $\mathcal{ALC}_x$ Syntax

Ontology Template: We denote $N_X = \{X_1, X_2, ..\}$ a set of concept name variables. We define language $\mathcal{ALC}_x$ by extending $\mathcal{ALC}$ $\mathcal{DL}$ as follows. A *concept template* is an expression defined by rules for $\mathcal{ALC}$ concepts (1) extended with the new rule $C \rightarrow X$, where $X \in N_X$. An *axiom template* is defined same way as $\mathcal{ALC}$ axiom in Section 3.1 with *concept templates* in place of *concepts*. Similarly, an *ontology template* is a finite set of *axiom templates*. We call axiom/ontology template *ground* if it does not contain any variable. A function $\Upsilon : N_X \rightarrow \mathcal{P}(N_A)$ is *variable domain function* – it maps each concept name variable to a subset of all concept names.

Ontology Template Set: An *ontology template set* is a set $\Lambda = \{T_i \mid 1 \leq i \leq n_\Lambda\}$ such that :

- each $T_i = \{t_{ij} \mid 1 \leq j \leq n_{T_i}\}$ is an *ontology template* that consists of *axiom templates* t_{ij}
- $var(t_{ij})$ is set of all variables occurring in the signature of axiom template t_{ij}
- $var(T_i) = \bigcup var(t_{ij})$ is set of all variables occurring in ontology template T_i
- $var(\Lambda) = \bigcup var(T_i)$ is set of all variables occurring in Λ

Recall the motivating example from Section 1 and consider the ontology of building components and materials that contains 4 axioms (we will use abbreviation of OM for concept $OrganicMaterial$ and VE for concept $VericalElement$ later in the text):

$$\begin{aligned} O_{CM} = \{&Pillar \sqsubseteq VerticalElement,\\ &VerticalElement \sqsubseteq Component,\\ &Wood \sqsubseteq OrganicMaterial,\\ &OrganicMaterial \sqsubseteq Material\} \end{aligned}$$

$\Lambda_{CM} = \{T_1, T_2\}$ is an ontology template set such that :

- $T_1 = \{\$COMP_1 \sqsubseteq \forall hasMaterial.\neg\$MAT_1\ \}$
- $T_2 = \{\_c : \$COMP_2 \sqcap \exists hasMaterial.\$MAT_2\ \}$

$\_c$ is an anonymous individual [28]. Ontology template T_1 states that all individuals of type $\$COMP_1$ must have

material that does not belong to concept $\$MAT_1$. Ontology template T_2 states that there exists some individual of type $\$COMP_2$ that has material of type $\$MAT_2$. Υ is variable domain function such that :

- $\Upsilon(\$COMP_1) = \Upsilon(\$COMP_2) = \{Pillar, VerticalElement, Component\}$
- $\Upsilon(\$MAT_1) = \Upsilon(\$MAT_2) = \{Wood, OrganicMaterial, Material\}$

Substitution Set: Intuitively we define notion of *substitution* σ and a set of *substitutions* $\Sigma = \{\sigma_1, \sigma_2, ..\}$. In context of some *ontology template* T these notions will help us to differentiate between one grounding of T and all possible groundings of T. In context of *ontology template set* these notions help us to differentiate between one grounding of all templates from Λ and all possible ontology templates groundings that define possible evolutions w.r.t. Λ. The concrete definitions are:

- $var \subseteq N_X$ – a set of variables
- $\sigma : var \rightarrow N_A$ is a *substitution* (*variable substitution function*) – a function from subset of concept variables to concept names. We will use notation: $\sigma = \{[X_1/A_1], ..., [X_n/A_n]\}$, where $[X_i/A_i]$ means that σ maps variable X_i to concept name A_i; $dom(\sigma)$ for domain of σ.
- σ is *ground substitution* w.r.t. T if $var(\sigma(T)) = \emptyset$. We call $\sigma(T)$ *grounding* of axiom template w.r.t. σ and define $template(\sigma) = T$. Similarly, *ground substitution* can be extended to Λ.
- σ_{wg} is the *weakest ground substitution* of σ w.r.t. T if $\forall\sigma'(\sigma'$ is ground substitution and an extension of $\sigma) \implies \sigma'(T) \models \sigma_{wg}(T)$
- Σ is a set of substitutions
- Σ is *ground substitution set* w.r.t. T if every $\sigma \in \Sigma$ is ground w.r.t. T. Similarly the definition can can be extended to Λ.

  As an example consider substitution $\sigma' = \{[\$COMP_1/Pillar]\}$ and ground substitutions $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ such that :

  - $\sigma_1 = \{[\$COMP_1/Pillar], [\$MAT_1/Wood]\}$
  - $\sigma_2 = \{[\$COMP_1/VE], [\$MAT_1/OM]\}$
  - $\sigma_3 = \{[\$COMP_2/Pillar], [\$MAT_2/OM]\}$
  - $\sigma_4 = \{[\$COMP_2/VE], [\$MAT_2/Wood]\}$

  σ' is not ground substitution w.r.t. T_1 as $dom(\sigma') = \{\$COMP_1\}$ does not contain $\$MAT_1$. σ_1, σ_2 are ground substitutions w.r.t. T_1, while σ_3, σ_4 are ground substitutions w.r.t. T_2. Thus $template(\sigma_1) = template(\sigma_2) = T_1$. $\{\sigma_1, \sigma_2\}$ and $\{\sigma_1, \sigma_3\}$ are examples of a ground substitution sets w.r.t. T_1 and Λ, respectively. $\{\sigma', \sigma_2\}$ is not ground substitution set w.r.t. T_1 or T_2.

Instantiation Set: In order to define possible extensions of a *core ontology* and an *ontology template set*, we define the notion of an *instantiation set*. Intuitively, any possible extension of the *core ontology* w.r.t. *ontology template set*, i.e. an ontology O' can be defined by one instantiation. O' is the result of applying the instantiation to the *core ontology* and *ontology template set*. Let O be a *core ontology*, Λ be a *ontology template set*, Σ_i a substitution set for each i:

- A substitution Σ is an *instantiation* w.r.t. Λ, if every $\sigma \in \Sigma$ is a ground substitution of an *ontology template* T from Λ such that $dom(\sigma) = var(T)$.
- $\mathcal{S} = \{\Sigma_1, \Sigma_2, ..\Sigma_i\}$ is an *instantiation set* w.r.t. Λ if each non-empty Σ_i is an instantiation w.r.t. Λ.
- cl($\mathcal{S}$) is a *closure of* $\mathcal{S}$ *under inclusion*, i.e. $cl(\mathcal{S}) = \{\Sigma' \mid \Sigma \in \mathcal{S} \wedge \Sigma' \subseteq \Sigma\}$. If $\mathcal{S} = cl(\mathcal{S})$ we say that instantiation $\mathcal{S}$ is *closed under inclusion*.
- $max_{\subseteq}(\mathcal{S})$ is a set of all maximal elements of $\mathcal{S}$ w.r.t. partially ordered set ordered by inclusion.
- An *instantiation set* is *complete* if it is a power-set of some substitution set, i.e. $\mathcal{S} = \mathcal{P}(\Sigma)$.

Any subset of $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ is an instantiation w.r.t. Λ, while substitution set $\{\sigma_1, (\sigma_2 \cup \sigma_3)\}$ is an example of a ground substitution set which is not an instantiation w.r.t. Λ. $\mathcal{S}' = \{\{\sigma_1, \sigma_2\}, \{\sigma_1, \sigma_3\}, \{\sigma_1\}\}$ is an example of instantiation set, while its closure under inclusion is a set $cl(\mathcal{S}') = \{\{\sigma_1, \sigma_2\}, \{\sigma_1, \sigma_3\}, \{\sigma_1\}, \{\sigma_2\}, \{\sigma_3\}, \emptyset\}$ and $max_{\subseteq}(\mathcal{S}') = \{\{\sigma_1, \sigma_2\}, \{\sigma_1, \sigma_3\}\}$

Instantiation Function: Intuitively, an *instantiation function* applies substitutions to relevant ontology templates. For ground substitutions it returns list of axioms, otherwise it returns list of partially substituted axiom templates. Let T be an *ontology template*, Λ an *ontology template set*, an *instantiation function* IF is defined as follows :

- $IF(\sigma, T) = \sigma(T)$
- $IF(\Sigma, \Lambda) = \{\alpha \mid \alpha \in IF(\sigma, T) \wedge (\sigma \in \Sigma) \wedge (T \in \Lambda) \wedge (dom(\sigma) \cap var(T) \neq \emptyset)\}$
- $IF(\mathcal{S}, \Lambda) = \bigcup_{\Sigma \in \mathcal{S}} IF(\Sigma, \Lambda)$

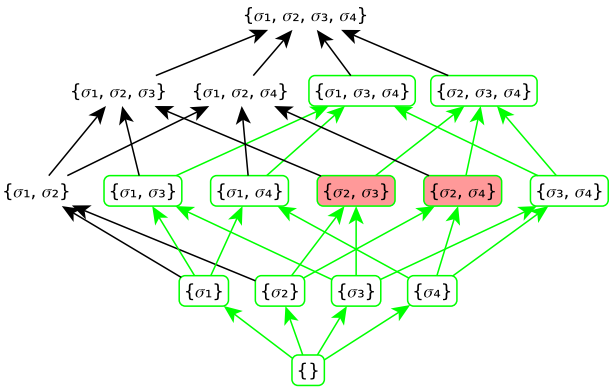


Figure 1: Complete modification graph w.r.t. a canonical evolution schema $\langle O, \Lambda, \mathcal{S}_\mathcal{G}, \mathcal{S}_\mathcal{V}\rangle$, green (red) sub-graph represents $\mathcal{S}_\mathcal{G}$ ($\mathcal{S}_\mathcal{V}$), respectively.

For example, $IF(\sigma', T_1) = \{Pillar \sqsubseteq \forall hasMaterial.\neg\$MAT_1\}$, while $IF(\sigma_1, T_1) = IF(\{\sigma_1\}, \Lambda) = IF(\{\{\sigma_1\}\}, \Lambda) = \{Pillar \sqsubseteq \forall hasMaterial.\neg Wood\}$

Modification Graph: Let $\mathcal{S}$ be some instantiation set representing all possible extensions of some *core ontology* O w.r.t. *ontology template set* Λ. A modification graph of $\mathcal{S}$ is an oriented graph where:

- each node represents one instantiation from $\mathcal{S}$,
- each edge connects instantiation from node representing Σ_i to node representing Σ_j if there exists not empty substitution σ such that $(\sigma \notin \Sigma_i) \wedge (\Sigma_i \cup \{\sigma\} = \Sigma_j)$.

We say that a modification graph is complete if it is a graph of a complete instantiation set. Figure 1 shows a modification graph for the complete instantiation set which is power-set of 4 grounding substitutions – $\sigma_1, \sigma_2, \sigma_3, \sigma_4$. In context of an ontology extension activity w.r.t. O and Λ, bottom node with label "{}" refers to a core ontology O, while top node with label "$\{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$" refers to ontology that was created by addition of 4 ontology template groundings – $O \cup IF(\{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}, \Lambda)$. Each edge in graph can be viewed as addition or removal (if opposite direction is considered) of one ontology template grounding.

Evolution Schema: For a template-based ontology extension activity we define an *evolution schema* as a 4-tuple $ES = \langle O, \Lambda, \mathcal{S}_\mathcal{G}, \mathcal{S}_\mathcal{V}\rangle$ such that

- O is a *core ontology*
- Λ is an *ontology template set*
- $\mathcal{S}_\mathcal{G}$ is *generating instantiation set* – an instantiation set that defines all possible template-based extensions of O w.r.t. Λ
- $\mathcal{S}_\mathcal{V}$ is *validating instantiation set* – an instantiation set that defines all minimal instantiations preserving inconsistency

In addition, we define :

- $\langle O, \Lambda, \mathcal{S}_\mathcal{G}, \mathcal{S}_\mathcal{V}\rangle$ is *canonical* if $\mathcal{S}_\mathcal{G}$ is closed under inclusion.
- $\mathcal{S}_C = \{\Sigma_\mathcal{G} \in \mathcal{S}_\mathcal{G} \mid \forall \Sigma_\mathcal{V} \in \mathcal{S}_\mathcal{V} : \Sigma_\mathcal{V} \not\subseteq \Sigma_\mathcal{G}\}$ is a *combined instantiation set* w.r.t. some evolution schema $\langle O, \Lambda, \mathcal{S}_\mathcal{G}, \mathcal{S}_\mathcal{V}\rangle$.

An example of a canonical evolution schema $\langle O, \Lambda, \mathcal{S}_\mathcal{G}, \mathcal{S}_\mathcal{V}\rangle$ is visualized in Figure 1. The generating instantiation set $\mathcal{S}_\mathcal{G}$ is visualized by green color, while validating instantiation set $\mathcal{S}_\mathcal{V}$ is visualized by red color. Note that, for any graph of instantiation set closed under inclusion (such as $\mathcal{S}_\mathcal{G}$) it holds that if it contains node representing substitution set Σ it as well contains complete sub-graph w.r.t. Σ.

4.2 $\mathcal{ALC}_x$ Semantics

Similarly to *MIPS* described in Section 3 but adapted to *ontology templates* we say that an instantiation Σ is *MIPS minimal inconsistency preserving set* w.r.t. an ontology O and an ontology template set Λ if following conditions holds :

- *inconsistency condition*, i.e. $(O \cup IF(\Sigma, \Lambda))$ is inconsistent
- *minimality condition*, i.e. $\forall \Sigma' \subset \Sigma \implies (O \cup IF(\Sigma', \Lambda))$ is consistent

Let $ES = \langle O, \Lambda, \mathcal{S}_\mathcal{G}, \mathcal{S}_\mathcal{V}\rangle$ be some evolution schema and $\mathcal{S}_C$ its combined instantiation set.

- An validating instantiation set $\mathcal{S}'_\mathcal{V}$ is *correct* w.r.t. evolution schema $\langle O, \Lambda, \mathcal{S}_\mathcal{G}, \mathcal{S}_\mathcal{V}\rangle$, if O is consistent and for each $\Sigma \in \mathcal{S}'_\mathcal{V}$, such that $\Sigma \in cl(\mathcal{S}_\mathcal{G}) : \Sigma$ is *MIPS* w.r.t. O and Λ.
- The evolution schema ES is *(necessarily) consistent* if $\mathcal{S}_\mathcal{V}$ is correct w.r.t. $\langle O, \Lambda, \mathcal{S}_\mathcal{G}, \emptyset\rangle$ and for every substitution set $\Sigma \in \mathcal{S}_C$ there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (O \cup IF(\Sigma, \Lambda))$.

Recall the example of the evolution schema $\langle O, \Lambda, \mathcal{S}_\mathcal{G}, \mathcal{S}_\mathcal{V}\rangle$ from Figure 1 as well as definitions of substitutions $\sigma_1, \sigma_2, \sigma_3, \sigma_4$. We will show that the evolution schema in the figure is consistent. First, it is easy to see that $\mathcal{S}_\mathcal{V} = \{\{\sigma_2, \sigma_3\}, \{\sigma_2, \sigma_4\}\}$ is correct w.r.t.

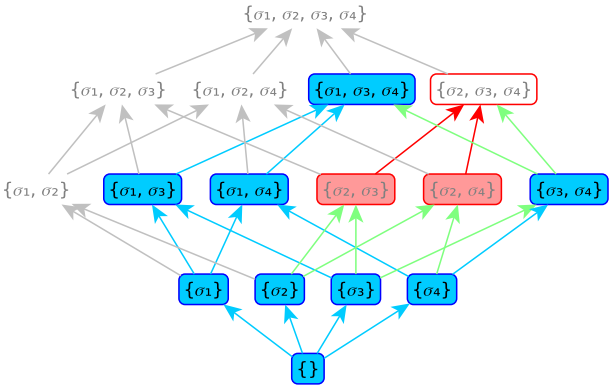


Figure 2: Complete modification graph w.r.t. a canonical evolution schema $\langle O, \Lambda, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{V}} \rangle$, sub-graph with red and blue nodes represents $\mathcal{S}_{\mathcal{G}}$, red sub-graph represents a set of inconsistent instantiations, blue sub-graph represents $\mathcal{S}_C$.

the evolution schema – ontologies $O \cup IF(\{\sigma_2, \sigma_3\}, \Lambda)$ and $O \cup IF(\{\sigma_2, \sigma_4\}, \Lambda)$ are both inconsistent, while ontologies O, $O \cup IF(\{\sigma_2\}, \Lambda)$, $O \cup IF(\{\sigma_3\}, \Lambda)$, and $O \cup IF(\{\sigma_4\}$ are consistent. Second, according to the provided definition $\mathcal{S}_C = \{\{\sigma_1, \sigma_3, \sigma_4\}, \{\sigma_1, \sigma_3\}, \{\sigma_1, \sigma_4\}, \{\sigma_3, \sigma_4\}, \{\sigma_1\}, \{\sigma_2\}, \{\sigma_3\}, \{\sigma_4\}, \{\}\}$. It can be shown that for each $\Sigma \in \mathcal{S}_C$ an ontology $O \cup IF(\Sigma, \Lambda)$ is consistent (i.e. has model). Figure 2 shows combined instantiation set $\mathcal{S}_C$ as blue sub-graph. Red sub-graph shows all inconsistent instantiations of the evolution. $\{\sigma_2, \sigma_3, \sigma_4\}$ is example of inconsistent instantiation which is not minimal. $max_{\subseteq}(\mathcal{S}_C)$ is set of all nodes that does not have parent in $\mathcal{S}_C$, i.e. $\{\{\sigma_1, \sigma_3, \sigma_4\}, \{\sigma_3, \sigma_4\}, \{\sigma_3\}\}$. Moreover, $\mathcal{S}_{\mathcal{V}}$ is *MIPS* w.r.t. O and Λ and note, that this is true for each consistent evolution.

5 Reasoning Service for Ontology Templates

OWL2 and many other $\mathcal{DL}$ based languages provide a set of useful services for querying, validating and debugging errors within an ontology. One of the most basic services is consistency checking of an ontology. If consistency check fails during manual development of the ontology, an ontology engineer typically starts appropriate debugging services to find the root cause of the problem and fix it.

Let's imagine, that an ontology engineer identifies some higher level patterns within the ontology and creates an *ontology template* in order to simplify his work. He assigns *domain* of each *variable* within the template which significantly reduces his time when filling in the templates. Unfortunately, he is still not satisfied as there are many dependent variables in the template. Some of them could be filled in automatically, others could have set of values significantly reduced based on already assigned variables of the form. He decides to create better instantiation function (not only based on variable domains) by providing some additional relationships between particular variables of the template. He creates new *ontology templates*, fills in some data based on them and runs consistency check, which fails. After he spends quite a long time trying to find out the ontology error, he decides to add *validation constraint* for *ontology templates*, so it won't happen again in such a situation. This again reduces variable values within the forms of *ontology templates*. He also adds non-technical textual description of *validation constraint* as he plans to provide some *ontology templates* forms to domain experts to populate the ontology. The only think he is missing is: 1) a way to find out whether the *validation constraint* is not too restrictive, i.e. it does not exclude some consistent *ontology template groundings*, 2) whether there are some other possible inconsistences that *ontology templates* can create. Points 1) and 2) can be answered by new reasoning service for *ontology templates* as follows.

Reasoning Services: Let $\langle O, \Lambda, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{V}} \rangle$ be an ontology evolution scheme and $\mathcal{S}'_{\mathcal{V}}$ an instantiation set:

- *validation constraint check* – take $\mathcal{S}'_{\mathcal{V}}$ and $\langle O, \Lambda, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{V}} \rangle$ as parameters and returns true if $\mathcal{S}'_{\mathcal{V}}$ is correct validation instantiation set w.r.t. evolution scheme $\langle O, \Lambda, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{V}} \rangle$. Otherwise it returns an ontology that proves incorrectness of the $\mathcal{S}'_{\mathcal{V}}$. Algorithm 1 provides description of the service.

- *evolution scheme consistency check* – take $\langle O, \Lambda, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{V}} \rangle$ as parameter returns true if $\langle O, \Lambda, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_{\mathcal{V}} \rangle$ is consistent. Otherwise it returns an ontology that proves either inconsistency of whole scheme or incorrectness of validation set $\mathcal{S}_{\mathcal{V}}$. Algorithm 3 provides description of the service.

In the following text we define three algorithms needed for reasoning services together with proof of it soundness and completeness. The algoritms have two outputs – an output value and global variable E which represents an error by tuple of a textual message and ontology that proves the error.

Algorithm 1 checks correctness of a validation constrains w.r.t. some evolution schema.

- *Termination* – The only looping structures are "foreach" constructs over finite immutable set, thus algorithm terminates.

- *Soundness* – If the algorithm returned *true*, thus the execution did not step inside of the code within the

lines *(2-4)*, *(7-9)* and *(12-14)* – we will make abbreviation *MISS()* to express this fact. The *MISS(2-4)* implies that the ontology O is consistent, the *MISS(7-9)* implies that for each $\Sigma \in (\mathcal{S}'_{\mathcal{V}} \cap cl(\mathcal{S}_{\mathcal{G}}))$ the *inconsistency criterion* holds. The *MISS(12-14)* implies that $\forall \sigma \in \Sigma : O \cup IF(\Sigma - \{\sigma\}, \Lambda)$ is consistent. For an arbitrary $\Sigma' \subset \Sigma$ it is true that $\Sigma' \subseteq \Sigma - \{\sigma\}$ for some σ. Thus $(O \cup IF(\Sigma', \Lambda)) \subseteq (O \cup IF(\Sigma - \{\sigma\}, \Lambda))$ and *minimality condition* implies from *monotonicity of entailment*[2] [2] in $\mathcal{ALC}$.

- *Completeness* – Let O be a consistent ontology, $\Sigma \in (\mathcal{S}'_{\mathcal{V}} \cap cl(\mathcal{S}_{\mathcal{G}}))$ and Σ is MIPS w.r.t. O and Λ. Algorithm returns *true* as *MISS(2-4)*, *MISS(7-9)* and *MISS(12-14)* is trivially guaranteed from preconditions.

- *Complexity* – Let O, Λ, $\mathcal{S}$ be an ontology, an ontology template set and an instantion set, respectively. We will use $l(O)$, $l(\Lambda)$, $l(\mathcal{S})$ to denote length (size) of their encoding within the algorithm. The upper bound complexity of ontology consistency (satisfiability) check of $\mathcal{ALC}$ is PSPACE [2]. Thus lines (2-4) are in PSPACE w.r.t. $l(O)$. Computation of the condition on line (5) is at most polynomial w.r.t. $\mathcal{S}_{\mathcal{G}}$ and $\mathcal{S}'_{\mathcal{V}}$. The size of O' and O'' is at most $l(O) + l(\Lambda) * l(\mathcal{S}'_{\mathcal{V}})$, thus polynomially bounded by $l(O)$, $l(\Lambda)$, and $l(\mathcal{S}'_{\mathcal{V}})$. Loop (5-14) is executed at most $l(\mathcal{S}'_{\mathcal{V}})$ times and its inner loop is thus executed at most $l(\mathcal{S}'_{\mathcal{V}})^2$ times. Size of O'' and O' is at most $l(O) + l(\Lambda) * l(\mathcal{S}'_{\mathcal{V}})$. Thus number of consistency checks within lines (5-14) as well as size of the checked ontology is bounded polynomially and therefore can be computed in PSPACE. From the above analysis it implies that Algorithm 1 uses input $l(\mathcal{S}_{\mathcal{G}})$ to compute only line (5) which can be done polynomially w.r.t. $l(\mathcal{S}_{\mathcal{G}})$ and $l(\mathcal{S}'_{\mathcal{V}})$. The algorithm is in PSPACE w.r.t. $l(O)$, $l(\Lambda)$, $l(\mathcal{S}'_{\mathcal{V}})$ and $l(\mathcal{S}_{\mathcal{G}})$.

Algorithm 2 computes an instantiation Σ' which is *MIPS* w.r.t. an ontology O, an ontology template set Λ, an instantiation Σ. It assumes to call two external functions : compute single MIPS of an ontology and compute hitting set, both explained in section 3. The algorithm works as follows :

- lines *(5-10)* define ontology $O'' = IF(\Sigma, \Lambda)$ and relation M that maps each axiom α such that $\alpha \notin O$ to a set of substitutions $\{\sigma_i\}$ that could lead to the grounded axiom α. σ_i represents some ground substitution of an ontology template from Λ,

[2]Monotonic logic such as $\mathcal{ALC}$ ensures that if O is consistent, than any subset O' is consistent as well.

Algorithm 1: Check validating constraint correctness

```
1  is_correct_VC (S'_V, ⟨O, Λ, S_G, S_V⟩) : boolean is
       Result: Returns true, if S'_V is correct validation
               instantiation set w.r.t. evolution scheme
               ⟨O, Λ, S_G, S_V⟩. Otherwise it returns
               false and an ontology that proves
               incorrectness of the S'_V.
2      if not is_consistent(O) then
3          E ← ⟨"core ontology is not consistent", O⟩
4          return false
5      foreach Σ ∈ (S'_V ∩ cl(S_G)) do
6          O' ← (O ∪ IF(Σ, Λ))
7          if is_consistent(O') then
8              E ← ⟨"validation constraint not
                   correct", O'⟩
9              return false
10         foreach σ ∈ Σ do
11             O'' ← (O ∪ IF(Σ − {σ}, Λ))
12             if not is_consistent(O'') then
13                 E ← ⟨"validation constraint is not
                       minimal", O''⟩
14                 return false
15     return true
```

Algorithm 2: Get single *MIPS* instantiation

```
1  get_single_MIPS_instantiation (O, Λ, Σ) : Σ' is
       Result: Returns an arbitrary instantiation Σ' ⊆ Σ
               that causes inconsistency w.r.t. O and is
               minimal with respect to template-based
               extension of O using Λ and Σ. In case
               where O is inconsistent or instantiation
               Σ does not lead to inconsistence, it
               returns ∅
2      if not is_consistent(O) then
3          E ← ⟨"core ontology is not consistent", O⟩
4          return ∅
5      M ← {}
6      O'' ← O
7      foreach σ ∈ Σ do
8          O' ← IF(σ, Λ) − O
9          M ← M ∪ {⟨α, σ⟩ | α ∈ O'}
10         O'' ← O'' ∪ O'
11     O_sMIPS ←get_single_MIPS(O'')
12     S_MIPS ← {{σ | ⟨α, σ⟩ ∈ M} | α ∈ O_sMIPS}
13     return get_single_minimal_HTS(S_MIPS)
```

- lines *(11-12)* compute single *MIPS* of ontology $O \cup IF(\Sigma, \Lambda)$ i.e. set of axioms O_{sMIPS} which are transformed to set of substitution sets $\mathcal{S}_{sMIPS}$ where each substitution represent some axiom from O_{sMIPS},

- line *(13)* compute minimal hitting set of $\mathcal{S}_{MIPS}$.

To prove the correctness of the algorithm let's assume that O is consistent, $O \cup IF(\Sigma, \Lambda)$ is inconsistent, but the algorithm outputs Σ' failing the *inconsistency condition*. From the monotonicity of entailment and fact that $O_{sMIPS} \subseteq O \cup IF(\Sigma, \Lambda)$ it implies that O_{sMIPS} must be consistent ontology which is contradiction. Let's assume that Σ' satisfies *inconsistency condition* but fails *minimality condition*. Thus there exists $\sigma \in \Sigma'$ and $\Sigma'' = \Sigma' - \{\sigma\}$ such that $O \cup IF(\Sigma'', \Lambda)$ is inconsistent. This implies that either computed hitting set is not minimal or axiom set returned by single mips algorithm is not minimal – this is constradiction, and thus we proved that Σ' must satisfy both *inconsistence* and *minimality* condition. *Termination* of the algorithm is satisfied because only looping structures are "foreach" constructs over finite immutable set.

Complexity of Algorithm 2 can be evaluated using function l that defines size of the encoding of the input as explained above. Lines (2-4) can be evaluated in PSPACE w.r.t. l(O). The size of O'' is polynomial w.r.t. $l(O)$, $l(\Lambda)$ and $l(\Sigma)$. Algorithm for single MIPS is PSPACE [23] and algorithm for minimal HTS is NP-complete [24] problem. Thus Algorithm 2 is PSPACE w.r.t. $l(O)$, $l(\Lambda)$, and $l(\Sigma)$.

Algorithm 3: Check evolution schema consistency

```
1  is_consistent_ES (⟨O, Λ, S_G, S_V⟩) : boolean is
       Result: Returns true, if evolution schema
               ⟨O, Λ, S_G, S_V⟩ is consistent. Otherwise
               it returns false and an ontology that
               proves its inconsistency.
2      if not is_correct_VC(S_V, ⟨O, Λ, S_G, ∅⟩) then
3          return false
4      S_C ← {Σ_G ∈ S_G | ∀Σ_V ∈ S_V : Σ_V ⊈ Σ_G}
5      foreach Σ ∈ max_⊆(S_C) do
6          Λ_sMIPS ←
           get_single_MIPS_instantiation(O, Λ, Σ)
7          if Λ_sMIPS ≠ ∅ then
8              E ← ⟨"evolution generate inconsistent
               ontology", O ∪ IF(Λ_sMIPS, Λ)⟩
9              return false
10     return true
```

Algorithm 3 checks consistency of evolution schema. It uses algorithm to compute single *MIPS* instantiation only to return meaningfull explanation if consistency check fails.

- *Termination* – implies from only one "foreach" constructs over finite set.

- *Soundness* – Let's assume that algoritm returned *true*. Thus from *MISS(2-3)* it implies that $\mathcal{S}_{\mathcal{V}}$ is correct and O is consistent. From lines *(5-9)* and *MISS(7-9)* it implies that for each $\Sigma \in max_{\subseteq}(\mathcal{S}_C)$ it implies that $O \cup IF(\Sigma, \Lambda)$ is consistent. For any $\Sigma' \in \mathcal{S}_C$ it holds that is subset of some Σ and again due to monotonicity it implies that $O \cup IF(\Sigma', \Lambda)$ is consistent, thus there exists an interpretation $\mathcal{I}$ that satisfies the ontology.

- *Correctness* – This assume we have interpretation $\mathcal{I}$ for each $\Sigma \in \mathcal{S}_C$ and that lines *(7-9)* are not executed. Thus algorithm returns *true*.

- *Complexity* – Let l be a function defining size of the encoding of the input as used above. Upper bound complexity of lines (2-3) is PSPACE w.r.t. $l(\mathcal{S}_{\mathcal{V}})$, $l(O)$, and $l(\Lambda)$. Computation of $\mathcal{S}_{\mathcal{G}}$ at line (4) is polynomial w.r.t. $l(\mathcal{S}_{\mathcal{G}})$ and $l(\mathcal{S}_{\mathcal{V}})$. Loop (5-9) is evaluated at most $l(\mathcal{S}_{\mathcal{G}})$ times. Single MIPS instantiations (line 6) is computed in PSPACE w.r.t. $l(O)$, $l(\Lambda)$, $l(\mathcal{S}_{\mathcal{G}})$) as explained above. Thus Algorithm 3 is in PSPACE w.r.t. $l(O)$, $l(\Lambda)$, $l(\mathcal{S}_{\mathcal{G}})$, and $l(\mathcal{S}_{\mathcal{V}})$.

6 Prototype Implementation

We created a prototype implementation of reasoning service over ontology templates as well as set of tools to manage templates within command-line interface [3].

It is written in Java and uses Pellet query engine [35] to evaluate SPARQL [17] and SPARQL-DL queries [34].

Ontology templates are encoded using syntax of "construct template" of SPARQL construct query [17]. Instantiation sets can be defined by a set of generating and validating SPARQL / SPARQL-DL queries . The definitions of ontology templates and instantiation sets are attached to an ontology using OWL2 compliant annotations.

There are 2 types of validating queries that we differentiate in implementation: *flat validating queries* – can validate ontology templates only with each other; *multidimensional validating queries* – can in addition formulate constraints including more groundings of one template. More formally, *flat validation query* represents instantiation set $\mathcal{S}_{\mathcal{V}}$ such that each $\Sigma \in \mathcal{S}_{\mathcal{V}}$ is

[3] http://kbss.felk.cvut.cz/web/portal/web/blaskmir/ontology-templates, cit. 10.8.2015

set of grounding substitutions of mutually different *ontology templates* (i.e. $\forall\sigma_i, \sigma_j \in \Sigma : (\sigma_i \neq \sigma_j) \implies template(\sigma_i) \neq template(\sigma_j)$). Examples of both types of the queries are shown in Section 7.

Flat validating queries and *generating queries* are SPARQL / SPARQL-DL select queries over the *core ontology*. Result of a query, i.e. variable bindings is after execution transformed into instantiation set. To evaluate *multidimensional validating queries* we introduced *Evolution schema ontology*, an OWL 2 ontology that describes concrete evolution schema. Within the ontology, each ground substitution of an ontology template is represented by unique individual together with its variable mappings. Thus query over evolution scheme ontology can answer questions such as "Which *ontology template groundings* (substitutions) has variable $\$COMP_1$ assigned to value *Component* ?". Instead of core ontology, a *multidimensional query* queries dataset consisting of *core ontology* and *evolution schema ontology*. This is implemented through "group graph pattern" [17] of SPARQL language.

7 Use Cases

We demonstrate usage of ontology templates in two different domains. First, within domain of cultural heritage, we show example of two *ontology templates* with generating and flat validating SPARQL-DL queries for them. Second, within domain of privacy protection we show one template and multidimensional validating SPARQL query for it.

7.1 MONDIS Project Use Case

Ontology templates are currently used within the research project MONDIS[4]. Aim of the project is to create a knowledge-based system for description of monuments, their damage analysis, intervention planning and prevention in the field of cultural heritage protection. Recall the example ontology from Section 4 about components and materials and their templates T_1, T_2:

- $T_1 = \{\$COMP_1 \sqsubseteq \forall hasMaterial.\neg\$MAT_1\ \}$
- $T_2 = \{\_c : \$COMP_2 \sqcap \exists hasMaterial.\$MAT_2\ \}$

The example ontology is proper fragment of Monument Damage Ontology [3, 6] developed within the project MONDIS. The ontology templates T_1 and T_2 were used by domain experts to collect general knowledge about sub-concepts of *Component* and *Material*.

The generating query used for the template T_1 was :

[4] http://www.mondis.cz, cit. 10.8.2015

$\$COMP_1$	$\$MAT_1$
Pillar	*Wood*
Pillar	*OrganicMaterial*
VerticalElement	*Wood*
VerticalElement	*OrganicMaterial*

Table 1: Result of generating query for T_1

```
SELECT $COMP1 $MAT1
WHERE {
    $COMP1 rdf:subClassOf Component .
    $MAT1 rdf:subClassOf Material .
    FILTER (! ($COMP1 = Component) )
    FILTER (! ($MAT1 = Material) )
}
}
```

The SPARQL-DL query over the *core ontology* returns variable bindings as shown in Table 1. Each row of the query result can be naturally transformed to instantiation with one substitution (e.g. second row of the table is $\{\sigma\}$ where $\sigma = \{[\$COMP_1/Pillar], [\$MAT_1/Wood]\}$. Same query, but with appropriate variable names was used for generating query of T_2.

One of the validating query was query :

```
SELECT $COMP1 $MAT1 $COMP2 $MAT2
WHERE {
    $COMP2 rdf:subClassOf $COMP1 .
    $MAT2 rdf:subClassOf $MAT1 .
}
```

Each row of the validating query result can be transformed into instantiation consisting of 2 substitutions $\{\sigma_i, \sigma_j\}$, e.g. :

- $\sigma_i = \{[\$COMP_1/Pillar], [\$MAT_1/Wood]\}$
- $\sigma_j = \{[\$COMP_2/Pillar], [\$MAT_2/OM]\}$

The generating query of T_1, generating query of T_2 and the validating query are actually not evaluated alone as the combined instantiation can be evaluated by one SPARQL query. On the other hand, if $\mathcal{S}_1$ and $\mathcal{S}_2$ are resulting instantiation sets of the generating queries w.r.t. T_1 and T_2, the canonical $\mathcal{S}_\mathcal{G}$ can be constructed by $\mathcal{P}(\mathcal{S}_1 \cup \mathcal{S}_2)$. Let $\mathcal{S}'$ be resulting instantiation set of the validation query. The validating instantiation set $\mathcal{S}_\mathcal{V}$ can be constructed by splitting each substitution within $\mathcal{S}'$ into two ground substitutions, i.e. $\mathcal{S}_\mathcal{V} = \{\{\sigma_1, \sigma_2\} \mid \sigma \in \mathcal{S}' \wedge \sigma_1 = \sigma|_{var(T_1)} \wedge \sigma_2 = \sigma|_{var(T_2)}\}$.

Within the MONDIS project, similar templates were created also between other top level concepts that are related to diagnosis and intervention of damages (i.e. concept $ManifestationOfDamage$, $Agent$, $Mechanism$, $Intervention$) [6]. In addition to this general knowledge,

some real object descriptions (e.g. about "north tower of Prague's castle and its crack") were collected using concept and role assertions.

7.2 P3P Use Case

Validating queries will be demonstrated on use case from domain of privacy protection. It will be also shown how two groundings of one ontology template can generate inconsistencies. P3P [26] is W3C recommendation for expressing service privacy practices. The practices are described by XML-based P3P policies in terms of data that will be collected, purposes for which the data will be used, a period how long the data will be held etc. In the paper [4], we proposed framework for detection and explanation of P3P policy inconsistencies using semantic technologies. The policies are transformed to an ontology and then validated by ontology consistency check and set of SPARQL-DL queries. This was implemented in P3P privacy policy editor [33]. Although P3P editor was able to find inconsistencies we were not able to provide user human-readable message to explain inconsistency as we did not know about them in advance. This can be done by framework described in this paper.

One of the ontology templates (T) used in our P3P framework is the following:

$$\begin{aligned}
\$\mathbf{DATA} &\sqsubseteq \exists \mathit{hasPurpose}.(\$\mathbf{PUR} \sqcap \\
&\quad \exists \mathit{hasRequirement}.\$\mathbf{REQ}) \\
\$\mathbf{DATA} &\sqsubseteq \forall \mathit{hasPurpose}.((\neg \$\mathbf{PUR}) \\
&\quad \sqcup \forall \mathit{hasRequirement}.\$\mathbf{REQ}) \\
\$\mathbf{DATA_TR} &\sqsubseteq \exists \mathit{hasPurpose}.\$\mathbf{PUR} \\
\$\mathbf{DATA_TR} &\sqsubseteq \forall \mathit{hasPurpose}.((\neg \$\mathbf{PUR}) \sqcup \\
&\quad \forall \mathit{hasRequirement}.(\mathit{Always} \sqcup \\
&\quad \$\mathbf{REQ_1} \sqcup \$\mathbf{REQ}))
\end{aligned}$$

The ontology template T is used to state that some data ($\$DATA$) such as "User's business contact information" is collected for some purpose ($\$PUR$) such as "Marketing of services or products". The $\$REQ$ variable indicates requirement to which extent is the purpose required for the service. The requirement can be one of 3 values : *opt-in* - the user has to affirmatively request usage of data for this purpose; *opt-out* - the data may be collected for this purpose unless the user requests otherwise; *always* - the data will be used for this purpose.

Variables $\$DATA\_TR$ and $\$REQ_1$ can be generated from variables $\$DATA$ and $\$REQ$, respectively. A validating query is expressed in SPARQL query, which can well describe the syntactic nature of generated dependencies. Validation query can be expressed using dataset of core ontology and evolution schema ontology as follows:

```
SELECT $IS1 $IS2 WHERE {

  # query to evolution schema ontology
  GRAPH p3p:evolutionSchema {
    $IS1 arg:DATA_TR $DATA_TR1 ;
         arg:PUR $PUR ;
         arg:REQ $REQ1 .
    $IS2 arg:DATA_TR $DATA_TR2 ;
         arg:PUR $PUR ;
         arg:REQ $REQ2 .
  }

  # query to core ontology
  GRAPH p3p: {
    $DATA_TR1
        (rdf:subClassOf)+ $DATA_TR2 .
  }
  FILTER(isStronger($REQ1, $REQ2))
}
```

The first part of the query is evaluated on evolution schema ontology. It selects all instantiations $\$IS_1$ and $\$IS_2$ such that their purposes are same value. $\$PUR$ is bound to this value. $\$DATA\_TR_1$ binds to "DATA_TR" variable of original template. Similarly, $\$DATA\_TR_2$, $\$REQ_1$ and $\$REQ_2$ are bound. The second part of the query is evaluated on the core ontology, i.e. the P3P ontology. *isStronger* is SPIN-based SPARQL function that relates requirements, it is true if used for arguments ⟨always, opt-out⟩, ⟨always, opt-in⟩, ⟨opt-out, opt-in⟩ and false otherwise. Transformation of the query to the validating instantiation set is straight-forward as each row of the query result is translated to one instantiation $\{\sigma_1, \sigma_2\}$ directly corresponding to values of $\$IS_1$ and $\$IS_2$.

An example that would cause the query to fire is : "User's business contact information" is collected for "Marketing of services or products" with requirement "always". "User's business telephone number" is collected for "Marketing of services or products" with requirement "opt-out".

We use the evolution schema ontology within TopBraid Composer [37] to edit concrete instantiations of templates, i.e. instead of creating ontology templates directly in P3P ontology we create only instantiations within the evolution schema ontology, which are then synchronized with the core ontology. It allows us to have specialized forms for each template as shown in Figure 3. In addition using the query above we can create SPIN constraint on instantiations that provides human-readable explanations of inconsistencies and fixes as it is shown in Figure 3 and Figure 4. The provided framework can be used with Free Edition of TopBraid Composer.

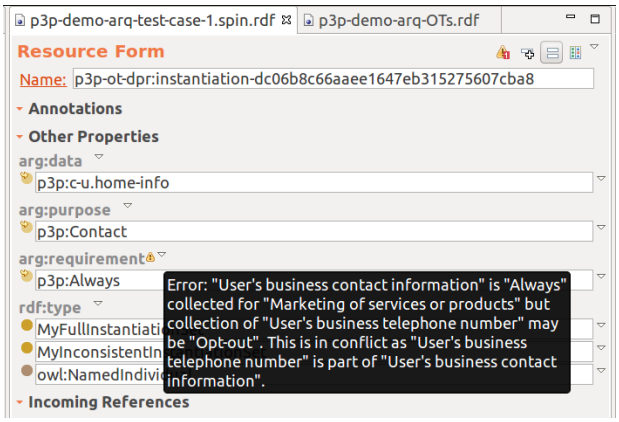


Figure 3: P3P error in TopBraid composer.

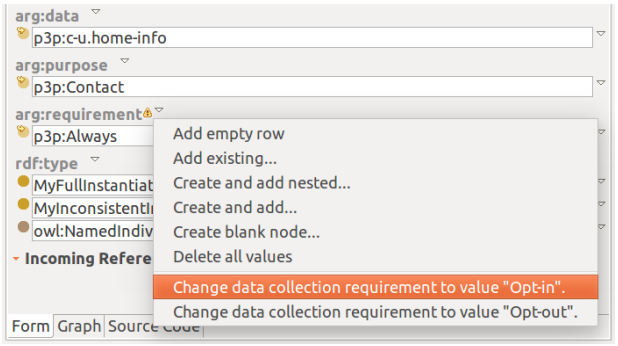


Figure 4: Suggestion for fix in TopBraid composer.

8 Summary and Conclusion

The paper introduced a formal technique for controlling ontology evolution by domain experts. The technique guarantees for them that – as long as the changes they make comply with the evolution templates, they will not result in an inconsistency.

The formal framework depends only on ontology consistency check and single MIPS procedure of the underlying $\mathcal{DL}$. Thus it can be easily extended to more expressive description logics, such as $SROIQ(D)$ [19] which is used in current ontology standard OWL 2. Moreover it seems straight-forward to extend the framework with role name variables and general concept variables.

The introduced technique has far-reaching impact on the process of ontology design and evolution. As our experience in the MONDIS project shows, domain experts feel more comfortable and less frustrated when they can predict the impact of changes they make. Another impact is the optimization of ontology engineering resources during the evolution phase.

The introduced research opened several interesting research topics. First, we introduced two cases of ontology evolution. In the future we would like to experiment with various ontology templates w.r.t. their usability for domain experts. Comparing to ontology design patterns that are meant for ontology engineers (in the same sense as software design patterns are meant for software engineers), ontology templates are meant for domain experts – thus, they have to be simpler, intuitive and have the formal guarantees introduced in this paper.

Another interesting research topic is the management of ontology templates themselves. Within a large ontology, many ontology templates can be defined, representing simple interaction schemes with the ontology for domain experts. For such cases, management and mutual interaction of such templates becomes an important topic. We will also deal with this problem in the next work.

Acknowledgements

This work has been supported by the grant "Defects in immovable cultural heritage objects: a knowledge-based system for analysis, intervention planning and prevention",
No. DF11P01OVV002 of the Ministry of Culture of the Czech Republic.

References

[1] L. Akroun, L. Nourine, and F. Toumani, "Reasoning in description logics with variables: preliminary results regarding the EL logic," in *Proceedings of the 28th International Workshop on Description Logics, Athens,Greece, June 7-10, 2015.*, ser. CEUR Workshop Proceedings, D. Calvanese and B. Konev, Eds., vol. 1350. CEUR-WS.org, 2015. [Online]. Available: http://ceur-ws.org/Vol-1350/paper-42.pdf

[2] F. Baader, *The description logic handbook: theory, implementation, and applications*. Cambridge university press, 2003.

[3] M. Blaško, R. Cacciotti, P. Křemen, and Z. Kouba, "Monument damage ontology," in *Progress in Cultural Heritage Preservation*. Springer, 2012, pp. 221–230.

[4] M. Blaško, P. Křemen, and Z. Kouba, "Privacy protection using semantic technologies," in *Proceedings of the 12th European Meeting on Cybernetics and Systems Research*, Vienna, 2009.

[5] R. Cacciotti, J. Valach, P. Kuneš, M. Čerňanskỳ, M. Blaško, and P. Křemen, "Monument damage information system (mondis), an ontological approach to cultural heritage documentation," *ISPRS Annals of the Photogrammetry, Remote Sensing*

and Spatial Information Sciences, vol. 2, no. 5, pp. 55–60, 2013.

[6] R. Cacciotti, M. Blaško, and J. Valach, "A diagnostic ontological model for damages to historical constructions," *Journal of Cultural Heritage*, vol. 16, no. 1, pp. 40–48, 2015.

[7] R. Cacciotti, J. Valach, P. Kuneš, and M. Čerňanský, "Knowledge-based system for documentation and mitigation of damages in historical structures," *CESB13-Central Europe towards sustainable building*, 2013.

[8] C. Chen and M. M. Matthews, "Extending description logic for reasoning about ontology evolution," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, 2007, pp. 452–456.

[9] S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, and A. Ragone, "Second-order description logics: Semantics, motivation, and a calculus," in *23rd International Workshop on Description Logics DL2010*, 2010, p. 67.

[10] Ó. Corcho, C. Roussey, L. M. V. Blázquez, and I. Pérez, "Pattern-based OWL ontology debugging guidelines," in *Proceedings of the Workshop on Ontology Patterns (WOP 2009) , collocated with the 8th International Semantic Web Conference (ISWC-2009), Washington D.C., USA, 25 October, 2009.*, ser. CEUR Workshop Proceedings, E. Blomqvist, K. Sandkuhl, F. Scharffe, and V. Svátek, Eds., vol. 516. CEUR-WS.org, 2009. [Online]. Available: http://ceur-ws.org/Vol-516/pap02.pdf

[11] G. De Giacomo, M. Lenzerini, and R. Rosati, "On higher-order description logics." in *Description Logics*, 2009.

[12] M. Egana, E. Antezana, and R. Stevens, "Transforming the axiomisation of ontologies: The ontology pre-processor language," *Proceedings of OWLED*, 2008.

[13] M. Fernandez-Lopez, A. Gomez-Perez, and N. Juristo, "Methontology: from ontological art towards ontological engineering," in *Proceedings of the AAAI97 Spring Symposium*, Stanford, USA, March 1997, pp. 33–40.

[14] D. M. Gabbay, R. Schmidt, and A. Szalas, *Second Order Quantifier Elimination: Foundations, Computational Aspects and Applications*. College Publications, 2008.

[15] A. Gangemi, A. Gomez-Perez, V. Presutti, and M. Suarez-Figueroa, "Towards a catalog of owl-based ontology design patterns," in *12th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2007*. Salamanca (Spain): AEPIA, 2007. [Online]. Available: http://www.neon-project.org/web-content/images/Publications/caepia-catalogpatterns-vfinal.pdf

[16] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquis.*, vol. 5, no. 2, pp. 199–220, Jun. 1993. [Online]. Available: http://dx.doi.org/10.1006/knac.1993.1008

[17] S. Harris, A. Seaborne, and E. Prudhommeaux, "Sparql 1.1 query language (2013)," *W3C Recommendation*, 2014.

[18] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, *A Practical Guide To Building OWL Ontologies With The Protege-OWL Plugin*, 1st ed., University of Manchester, 2004. [Online]. Available: http://home.skku.edu/~samoh/class/sw/ProtegeOWLTutorial.pdf

[19] I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible sroiq." *KR*, vol. 6, pp. 57–67, 2006.

[20] L. Iannone, A. Rector, and R. Stevens, "Embedding knowledge patterns into owl," in *The Semantic Web: Research and Applications*. Springer, 2009, pp. 218–232.

[21] A. Kalyanpur, "Debugging and repair of owl ontologies," Ph.D. dissertation, University of Maryland at College Park, College Park, MD, USA, 2006, aAI3222483.

[22] A. Kalyanpur, B. Parsia, E. Sirin, and B. C. Grau, "Repairing unsatisfiable concepts in owl ontologies." in *ESWC*, ser. Lecture Notes in Computer Science, Y. Sure and J. Domingue, Eds., vol. 4011. Springer, 2006, pp. 170–184.

[23] A. Kalyanpur, B. Parsia, E. Sirin, and J. Hendler, "Debugging unsatisfiable classes in owl ontologies," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 4, 2005. [Online]. Available: http://www.websemanticsjournal.org/index.php/ps/article/view/77

[24] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, ser. The IBM Research Symposia Series, R. Miller, J. Thatcher, and J. Bohlinger, Eds. Springer US, 1972, pp. 85–103. [Online]. Available: http://dx.doi.org/10.1007/978-1-4684-2001-2_9

[25] H. Knublauch, J. Hendler, and K. Idehen, "Spin – overview and motivation," http://www.w3.org/Submission/2011/

SUBM-spin-overview-20110222/, 2011, accessed: 2015-09-14.

[26] M. Marchiori, "The platform for privacy preferences 1.0 (P3P1.0) specification," W3C, W3C Recommendation, Apr. 2002, http://www.w3.org/TR/2002/REC-P3P-20020416/.

[27] B. Motik, "On the properties of metamodeling in owl," *Journal of Logic and Computation*, vol. 17, no. 4, pp. 617–637, 2007.

[28] B. Motik, P. F. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler *et al.*, "Owl 2 web ontology language: Structural specification and functional-style syntax," *W3C recommendation*, vol. 27, no. 65, p. 159, 2009.

[29] J. Z. Pan and I. Horrocks, "Owl fa: a metamodeling extension of owl d," in *Proceedings of the 15th international conference on World Wide Web*. ACM, 2006, pp. 1065–1066.

[30] P. Patel-Schneider, B. Parsia, and B. Motik, "OWL 2 web ontology language structural specification and functional-style syntax (second edition)," W3C, W3C Recommendation, Dec. 2012, http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/.

[31] R. Reiter, "A theory of diagnosis from first principles," *Artificial intelligence*, vol. 32, no. 1, pp. 57–95, 1987.

[32] M. Schmidt-Schauß and G. Smolka, "Attributive concept descriptions with complements," *Artif. Intell.*, vol. 48, no. 1, pp. 1–26, 1991. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(91)90078-X

[33] P. Schneider, "P3p editor based on semantic technologies," Master's thesis, Czech Technical University in Prague, Prague, Czech Republic, 2012.

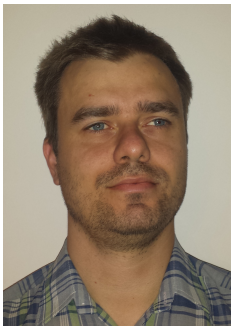
[34] E. Sirin and B. Parsia, "Sparql-dl: Sparql query for owl-dl." in *OWLED*, vol. 258, 2007.

[35] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semant.*, vol. 5, no. 2, pp. 51–53, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2007.03.004

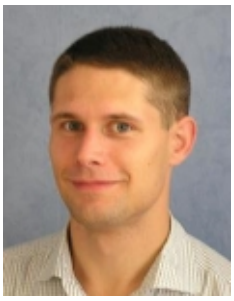
[36] L. Stojanovic, "Methods and tools for ontology evolution," Ph.D. dissertation, Karlsruhe Institute of Technology, 2004. [Online]. Available: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000003270

[37] TopQuadrant, "Tobbraid composer," http://www.topquadrant.com/tools, accessed: 2015-09-14.

Author Biographies



Miroslav Blaško is PhD student in the field of artifical intelligence and biocybernetics from the Czech Technical University in Prague, Czech Republic. His research topics are ontology development, error explanation and query answering that he applied in medicine, privacy protection and cultural heritage domain.



Dr. Petr Křemen received his Ph.D. degree in artifical intelligence and biocybernetics from the Czech Technical University in Prague, Czech Republic. He leads a research team at the Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University, Prague in the field of ontology-based information systems, ontology development, ontology comparison, error explanation and query answering. He is an author of more than 30 peer-reviewed articles, mainly on international fora.

Dr. Zdeněk Kouba is an Associate Professor with the Czech Technical University, Prague, Czech Republic. His research interests include design of information and knowledge based systems, data modeling and knowledge representation. He is a head of Knowledge-based and Software Systems group in the Department of Cybernetics at the Faculty of Electrical Engineering, Czech Technical University. His team has participated in six international research projects of the European Commission in past 20 years.