Proceedings of the Second International Workshop on Debugging Ontologies and Ontology Mappings - WoDOOM13

Montpellier, France
May 27, 2013.

Edited by:
Patrick Lambrix
Guilin Qi
Matthew Horridge
Bijan Parsia

Preface

Developing ontologies is not an easy task and, as the ontologies grow in size, they are likely to show a number of defects. Such ontologies, although often useful, also lead to problems when used in semantically-enabled applications. Wrong conclusions may be derived or valid conclusions may be missed. Defects in ontologies can take different forms. Syntactic defects are usually easy to find and to resolve. Defects regarding style include such things as unintended redundancy. More interesting and severe defects are the modeling defects which require domain knowledge to detect and resolve such as defects in the structure, and semantic defects such as unsatisfiable concepts and inconsistent ontologies. Further, during the recent years more and more mappings between ontologies with overlapping information have been generated, e.g. using ontology alignment systems, thereby connecting the ontologies in ontology networks. This has led to a new opportunity to deal with defects as the mappings and other ontologies in the network may be used in the debugging of a particular ontology in the network. It also has introduced a new difficulty as the mappings may not always be correct and need to be debugged themselves.

The WoDOOM series deals with these issues. This volume contains the proceedings of its second edition: WoDOOM13 - Second International Workshop on Debugging Ontologies and Ontology Mappings held on May 27, 2013 in Montpellier, France. WoDOOM13 was an ESWC 2013 (10th Extended Semantic Web Conference) workshop.

In his excellent invited talk, Heiner Stuckenschmidt proposed approaches for debugging weighted ontologies. In this generalization of the classical debugging problem, axioms in the ontology to be debugged have weights assigned and the task is to remove axioms from this set such that the resulting model is consistent and the sum of weights is maximal. Further, there were presentations of six full papers. The topics included both detection and repair of defects. Several papers used patterns for the detection. Regarding repairing wrong information, one paper proposed a method for reformulating axioms with the aim to retain as much information as possible. Another paper formalized the repairing of missing information in ontologies as a new abductive reasoning problem. Finally, a recently started EU project was presented in which ontology and mapping management is one of the core components. Two of the papers were selected for republication in the ESWC 2013 post-proceedings.

The editors would like to thank the Program Committee for their work in enabling the timely selection of papers for inclusion in the proceedings. We also appreciate our cooperation with EasyChair as well as our publisher CEUR Workshop Proceedings.

May 2013

Patrick Lambrix
Guilin Qi
Matthew Horridge
Bijan Parsia

Workshop Organization

Workshop Organizers

Patrick Lambrix	Linköping University, Sweden
Guilin Qi	Southeast University, China
Matthew Horridge	Stanford University, USA
Bijan Parsia	University of Manchester, UK

Program Committee

Samantha Bail	University of Manchester, UK
Bernardo Cuenca Grau	University of Oxford, UK
Jianfeng Du	Guangdong University of Foreign Studies, China
Peter Haase	fluid Operations, Germany
Aidan Hogan	Digital Enterprise Research Institute, Ireland
Matthew Horridge	Stanford University, USA
Maria Keet	University of KwaZulu-Natal, South Africa
Patrick Lambrix	Linköping University, Sweden
Yue Ma	TU Dresden, Germany
Christian Meilicke	University of Mannheim, Germany
Bijan Parsia	University of Manchester, UK
Rafael Peñaloza	TU Dresden, Germany
Guilin Qi	Southeast University, China
Ulrike Sattler	University of Manchester, UK
Stefan Schlobach	Vrije Universiteit Amsterdam, The Netherlands
Bariş Sertkaya	SAP Research Dresden, Germany
Kostyantyn Shchekotykhin	University Klagenfurt
Kewen Wang	Griffith University, Australia
Peng Wang	Southeast University, China
Renata Wassermann	University of Sao Paulo, Brazil
Fang Wei-Kleiner	Linköping University, Sweden

Table of Contents

Invited talk

Papers

Papers marked with (*) were selected for republication in the ESWC 2013 post-proceedings.

Debugging Weighted Ontologies

Heiner Stuckenschmidt

University of Mannheim, Germany

Abstract. We present our work on debugging weighted ontologies. We define this problem as computing a consistent subontology with a maximal sum of axiom weights. We present a reformulation of the problem as finding the most probable consistent ontology according to a log-linear model and show how existing methods from probabilistic reasoning can be adapted to our problem. We close with a discussion of the possible application of weighted ontology debugging to web scale information extraction.

1 Motivation

Probably the most often quoted advantage of logic-based ontologies are the possibility to check the model for different kinds of logical inconsistencies as possible symptoms of modeling errors. Since the work of Schlohbach and Cornet [19] many researchers have investigated the task of debugging description-logic ontologies, which dies not only include the detection of logical inconsistency, but also the identifying minimal sets of axioms causing it and removing axioms from the ontology to make it consistent again (e.g. [18, 16, 6, 8]).

While computing the cause of an inconsistency is relatively well understood and established techniques from diagnostic reasoning like the hitting set algorithm have been successfully applied and adapted to the problem of debugging ontologies, the decision which axioms to discard to retain consistency is still a largely unsolved problem. The classical solution used for instance in the field of believe revision is principle of minimal change that prefers solutions that remove the least number of axioms (compare e.g. [21]. While this approach has theoretical merits, it is not adequate for practical applications. For certain special cases such as debugging ontology mappings we can even observe that the principle of minimal change will remove correct axioms in most cases leaving incorrect ones in. As a consequence, researchers have focused on interactive debugging methods where a human user decides which axioms to remove while being supported by the debugging system [8, 10].

While interactive repair of ontologies is feasible when ontologies are rather small, more recently researchers get interested in debugging ontologies that have been automatically created from text or data sources. The resulting models are typically quite big and contain a high number of inconsistencies. While many classical debugging tools already have trouble in more classical settings as we have shown in our study on the practical applicability of debugging [20], using these tools on sets of automatically generated axioms turns out to be a hopeless endeavor.

In this paper, we summarize work on a new approach to ontology debugging that can be seen as a generalization of classical ontology debugging and that is also better suited for the task of debugging large, highly inconsistent models. Our approach is

based on the idea that axioms in the ontology to be debugged have weights assigned and the task is to remove axioms from this set such that the resulting model is consistent and the sum of weights is maximal. The second part of this definition provides us with an unambiguous criterion for selecting axioms to remove. Further, this definition of debugging weighted ontologies is equivalent to computing the most likely model in log-linear probabilistic models. We can use this correspondence to apply scalable inference mechanisms from the area of statistical relational learning to the task of ontology debugging.

The remainder of the paper is structured as follows: we first introduce a rather generic model of weighted ontologies that applies to different logical formalisms including light weight description logics and explain the relation to log linear models. In the second part of the paper, we discuss different different algorithms for debugging weighted ontologies based on linear integer programming and on Markov Chain Monte Carlo Sampling. We also discuss approach for scaling up these algorithms by distribution and parallel processing. We close with a discussion of open issues and future work.

2 Weighted Ontologies

2.1 Ontologies

We use a rather abstract ontology model that regards an ontology as a set of Axioms $\mathcal{O} = \{A_1, \cdots, A_n\}$. We represent axioms as predicates over constants representing classes, relations and instances. Existing representations of ontologies can be transferred into this representation by first normalizing the logical representation, eventually introducing new concept constants and then translating normalized axioms into literals. A complete translation for the logic $\mathcal{EL}^+$ can be found in [13]. The following example shows our representation of an ontology talking about philosophers and celestial objects:

$$A_1 : type(Pluto, Philosopher) \tag{1}$$
$$A_2 : related(born-in, Pluto, Athens) \tag{2}$$
$$A_3 : domain(born-in, Person) \tag{3}$$
$$A_4 : type(Pluto, DwarfPlanet) \tag{4}$$
$$A_5 : subconcept(Philosopher, Person) \tag{5}$$
$$A_6 : subconcept(Planet, CelestialObject) \tag{6}$$
$$A_7 : subconcept(DwarfPlanet, Planet) \tag{7}$$
$$A_8 : disjoint(CelestialObject, Person) \tag{8}$$

Our model further assumes the existence of an entailment relation $\models$ between sets of Axioms. Often, the entailment relation can be computed using a finite set of derivation rules. This observation corresponds to the investigation of consequence driven reasoning for description logics. In particular, for any description logic supporting consequence driven reasoning, we can compute the entailment relation between sets of using

derivation rules over the predicate representation. A correct and complete set of inference rules for $\mathcal{EL}^+$ can be found in [13]. For our example, we assume the following (incomplete set of) derivation rules for computing the entailment relation.

$$type(X, C) \wedge subclass(C, D) \vdash type(X, D) \quad (9)$$

$$subclass(C, D) \wedge subclass(D, E) \vdash subclass(C, E) \quad (10)$$

$$domain(R, C) \wedge related(X, R, Y) \vdash type(X, C) \quad (11)$$

$$type(X, C) \wedge type(X, D) \wedge disjoint(C, D) \vdash \bot \quad (12)$$

$$subclass(C, D) \wedge disjoint(C, D) \vdash \bot \quad (13)$$

We include the $\bot$ symbol for representing conflicts in the ontology. Abusing notation, we use $\bot$ for any kind of conflicts we want to exclude from the model. Concerning classical debugging the operator can be used to determine the existence of a logical inconsistency as well as incoherent classes in the same framework. We could also include domain-specific types of inconsistencies and detect them using the same algorithms as for the logical inconsistencies.

I our model, the task of ontology debugging can now be defined as finding a minimal subontology $\mathcal{O}' \subseteq \mathcal{O}$ such that $\mathcal{O}' \not\models \bot$ and there is no other subontology $\mathcal{O}' \subseteq \mathcal{O}''$ with this property. In our example such a sub-ontology can be generated by either removing axiom 1 and 2 or any of the axioms 3 to 8.

2.2 Weighted Axioms and log-linear Models

In our work, we consider cases, where not all axioms in an ontology have the same status, but some are preferred over others. We model this preference by a simple weight function $w : \mathcal{O} \rightarrow R \cup \{\infty\}$ where R denotes all real numbers and the weight function maps each axiom of an ontology either on a real number or on $\{\infty\}$ if the axiom should not be removed in any case. In the presence of a weight function, the notion of debugging is slightly changed. It can now be phrased as the task of finding a sub ontology $\mathcal{O}' \subseteq \mathcal{O}$ such that $\mathcal{O}' \not\models \bot$ and the sum of the weights in the axioms is maximal:

$$\sum_{A_i \in \mathcal{O}'} w(A_i) \geq \sum_{A_j \in \mathcal{O}''} w(A_j), \forall \mathcal{O}'' \subseteq \mathcal{O}$$

Let us assume that the first two axioms in our example have been automatically extracted while the other statements have been manually created by an expert. We could model this situation by assigning a lower weight to the first two axioms and higher weights to the other statements to indicate that we have more trust in the manually created parts of the model. So we might define $w(A_i) = 2, i \in \{1, 2\}$ and $w(A_i) = 5, i < 2$. In this case the only debugging of the resulting weighted ontology is $\mathcal{O}' = \{A_3, \cdots, A_8\}$ with a weight-sum of 30, whereas all other possible debuggings have at most a weight sum of 29.

In our work, we exploit the duality of this definition of debugging with log-linear models - probabilistic models where the a priori probabilities are given in terms of real-valued weights that are treated as logarithms of the actual probability. Thus, computing

the joint probability of independent events is done by summing up the weights instead of multiplying the probabilities. This means that the ontology with the highest weight sum is the most probable ontology according to a log-linear model over the weights of the axioms. In the case of only positive weights as in our example, the most probable ontology is always the one that contains all axioms. If we, however, force the probability of any sub ontology $\mathcal{O}' \models \bot$ to be zero, computing the most probable ontology turns out to be equivalent to computing a debugging as defined above. In particular, we define the probability of a subontology as follows:

$$P(\mathcal{O}') = \begin{cases} \frac{1}{Z} \exp\left(\sum_{\{A_i \in \mathcal{O}'\}} w(A_i)\right) & \text{if } \mathcal{O}' \not\models \bot \\ 0 & \text{otherwise} \end{cases}$$

Using this definition, debuggings of an ontology are simply the results of $\mathrm{argmax}_{\mathcal{O}' \subseteq \mathcal{O}}(P(\mathcal{O}'))$.

3 Debugging Algorithms

Actually computing debuggings is quite challenging is requires a combination of logical (for checking whether $\bot$ follows from a subontology) and probabilitic (for computing the probability of a model) inference. It turns out that naive approaches although they work for some special cases such as debugging alignments between small ontologies [9], fail to scale up to real world ontologies. At this point, we directly benefit from the above explained duality of debugging and inference in log-linear models, because we can build upon existing work in the area of probabilistic inference and design reasoning methods that scale up to very large (weighted) knowledge bases.

In the following, we describe two directions of work on algorithms for efficient debugging of weighted ontologies: the first one is based on a translation into an optimization problem that can be computed by solving a linear integer program. This work has already successfully been implemented in the ELOG reasoning system [1] develop at the university of Mannheim and is ready to use with OWL ontologies that have weights assigned as annotation properties [14]. The second direction is based on the idea of Sampling-based approximate inference that has the potential to scale to very large models. This work, that is based on Markov Chain Monte Carlo Sampling of ontologies has so far mostly been investigated on a theoretical level. First experiments have been made that show the potential of the method, but so far no stable reasoner is available.

3.1 Exact Inference using Linear Integer Programming

The first direction of work is based on the simple observation, that computing the most probable model can be phrased as an optimization problem and represented in terms of a linear integer programm. A linear integer program consists of an objective function that consists of the sum of integer variables with weights that has to be maximized. Further, side-conditions on the values of the variables can be stated in terms of linear inequalities over the variables. As we are interested in the presence or absence of axioms in an

[1] http://code.google.com/p/elog-reasoner/

ontology, we only consider Variables that have values from $\{0, 1\}$. A simple example of a linear integer program is $\mathbf{maximize}\, 0.6x_1 + 1.0x_2 + 0.5x_3$, $\mathbf{subjectto}\, x_1 + x_2 + x_3 \leq 1.2$. The solution of the example is: $x_1 = 1, x_2 = 0, x_3 = 1$. Instantiating the variables in the objective function with these values results in an objective value of 1.1.

The main task is now to find an optimal encoding of the problem into an integer linear programme. Riedel has proposed such a translation as a basis for efficient inferene in Markov logic [17]. As our representation of axioms as predicates and well as the corresponding inference rules can be represented as a Markov logic model, we can use the proposed translation as a basis for solving our problem. In particular, we can use the following steps for translating an ontology and the corresponding deduction rules into a linear integer programme:

1. replace non ground formulas with all possible groundings
2. Convert the resulting propositional knowledge base to conjunctive normal form
3. For each ground clause g determine positive $L^+(g)$ and negative $L^-(g)$ literals.
4. Determine the objective function as sum over all ground clause variable z_g and their weights
5. For each ground clause with weight $\neq \infty$ add the following constraints:

$$\sum_{l \in L^+(g)} x_l + \sum_{l \in L^(g)} (1 - x_l) \geq z_g$$

$$x_l \leq z_g, \forall l \in L^+(g)$$

$$(1 - x_l) \leq z_g, \forall l \in L^-(g)$$

6. For each ground clause with weight $= \infty$ add the following constraint

$$\sum_{l \in L^+(g)} x_l + \sum_{l \in L^(g)} (1 - x_l) \geq 1$$

7. Add the constraint $x_\perp = 0$ to enforce that $\perp$ is excluded from the model.

The solution of the corresponding debugging problem can be read from the solution of the linear integer programme. Each axiom in the ontology corresponds to a variable in the objective function, the solution of the debugging problem is the ontology that results from excluding all axioms from the model whose value is 0 in the objective function.

In our work [15] we have further optimized the translation procedure by translating clauses that share literals into a single constraint with counting variables. This approach has been shown to deliver a significant improvement for models with a high number of constants as it exploits symmetries in the resulting ground formulas to avoid repeated computations.

3.2 Approximate Inference using Markov-Chain Monte Carlo Sampling

While the ILP-based approach described above works well for medium sized knowledge-based, it runs into problems for very large models. In particular, if we think about using the methods on web scale, we quickly recognize that an optimal approach like the one described above is bound to fail. In such situations, where optimal algorithms fail, we can still use approximate inference methods for probabilistic models. A class of approximate inference methods that turned out to apply to our problem is Markov Chain Monte Carlo Sampling. In particular, we can adapt methods for sampling in dependent node sets from hypergraphs for our problem. For this purpose, we interpret an ontology as a hypergraph, where every axiom is a node in the hypergraph and nodes are connected by a hyperedge iff they form a diagnosis (i.e. a minimal set of axioms from which $\bot$ follows). A debugging of the ontology then corresponds to finding a maximal independent node set with respect to the weights of the axioms. Such an independent node set can now be determined by a Markov chain [7]. In [12] we proposed the following Markov Chain for computing weight-optimal debuggings in the sense of this paper.

A markov chain is a stochastic process with discrete time steps that is memoryless in the sense that its state at time t only depends on the state in t-1. Markov Chain Monte Carlo Methods are a class of algorithms that sample a probability distribution by constructing a Markov Chain that converges towards the desired distribution. We construct a Markov chain whose states are axiom subsets of the original ontology. It starts with an empty set of axioms and converges towards a state that corresponds to the weight optimal debugging of the ontology. Let $X^{(t)}$ be the state of the Markov Chain at time t, the state of the chain at time t+1 is computed as follows:

- chose and Axiom A uniformly at random
- if A is in $X^{(t)}$ then remove it with probability $\frac{1}{(\exp(w(A))-1)}$
- if A is not in $X^{(t)}$ and it is not in any diagnosis than add it with probability $\frac{\exp(w(A))}{(1+\exp(w(A)))}$
- if A is not in $X^{(t)}$ and it is in a diagnosis, then choose an other axiom from that diagnosis as random and replace it with A with probability $\frac{(m-1)\exp(w(A))}{(2m\exp w(A)-1)}$

First experiments on the PROSPERA Dataset [11] indicate that the method works well also on very large datasets that cannot be handled by optimal algorithms any more.

4 Conclusion: Debugging the Web

In this paper, we discussed the problem of debugging weighted ontologies. The problem can be seen as a generalization of ontology debugging where we have additional information about axiom preference in terms of weights assigned to axioms that can be used to compute a consistent ontology with a maximal sum of weights. We discussed the relation to computing the most probable consistent ontology using log-linear models and showed how we can exploit existing work from the field of probabilistic inference to efficiently compute debuggings. We believe that this method has a lot of potential and a lot of applications, in particular with respect to improving the results of web-scale information extraction.

As already indicated in the previous sections, our aim is to scale up the methods as far as possible. The ultimate goal is to address the web as a source of universal knowledge about the world. Recently a number of large scale knowledge extraction projects have been launched including NELL [2] TEXTRUNNER [3] and Knowitnow [1]. These projects extract more or less accurate facts from webpages building large knowledge bases about the world. Despite the use of high end extarctoin methods, the resulting models still contain mistakes and contradictions that need to be resolved to have a reliable model of world knowledge. In principle, our methods are able to integrate the results of these systems into a single, non conflicting model. For this purpose, however, we have to solve two problems: the first is to make our methods work on the scale of millions of facts as provided by these projects, further we have to model knowledge about conflicts between different facts in terms of a background ontology. While the first one is currently being addressed in terms of implementing the above mentioned sampling approach on a hadoop-based distributed infrastructure, we address the second problem by aligning the results of the extraction projects to the dbpedia ontology by matching objects and relations. If successful, we can use existing work on enriching the DBpedia ontology [5, 4] to determine logical inconsistencies.

Acknowledgement

The work summarized in this abstract has been joint work with Christian Meilicke, Mathias Niepert and Jan Noessner

References

1. Michael J. Cafarella, Doug Downey, Stephen Soderland, and Oren Etzioni. Knowitnow: Fast, scalable information extraction from the web. In *Proceedings of the Conference on Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP)*, 2005.
2. A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E.R. Hruschka Jr, and T.M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the 24th Conference on Artificial Intelligence (AAAI)*, page 13061313, 2010.
3. O. Etzioni, M. Banko, S. Soderland, and D.S. Weld. Open information extraction from the web. *Communications of the ACM*, 51(12):68–74, 2008.
4. Daniel Fleischhacker and Johanna Vlker. Inductive learning of disjointness axioms. In *On the Move to Meaningful Internet Systems: OTM 2011 : Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011*, Lecture Notes in Computer Science, pages 680–697. Springer, 2011.
5. Daniel Fleischhacker, Johanna Vlker, and Heiner Stuckenschmidt. Mining rdf data for property axioms. In *On the Move to Meaningful Internet Systems: OTM 2012 : Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012*, Lecture notes in computer science, pages 718–735. Springer, 2012.
6. Gerhard Friedrich and Kostyantyn Shchekotykhin. A general diagnosis method for ontologies. In *Proceedings of 4th International Conference on Semantic Web (ISWC?05)*, pages 232–246, Galway, Ireland, 2005.

7. M. Jerrum and A. Sinclair. The markov chain monte carlo method: an approach to approximate counting and integration. In *Approximation algorithms for NP-hard problems*, pages 482–520. PWS Publishing, 1996.
8. Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca-Grau. Repairing unsatisfiable concepts in owl ontologies. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006*, volume 4011 of *Lecture Notes in Computer Science*, pages 170–184, Budva, Montenegro, June 2006.
9. Christian Meilicke and Heiner Stuckenschmidt. Applying logical constraints to ontology matching. In *KI 2007: Advances in Artificial Intelligence : 30th Annual German Conference on AI*, 2007.
10. Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Supporting manual mapping revision using logical reasoning. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, Chicago, Illinois, USA, July 2008. AAAI Press.
11. N. Nakashole, M. Theobald, and G. Weikum. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of the 4th International Conference on Web Search and Data Mining (WSDM)*, pages 227–236, 2011.
12. Mathias Niepert, Christian Meilicke, and Heiner Stuckenschmidt. Towards distributed mcmc inference in probabilistic knowledge bases. In *NAACL-HLT 2012 Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction (AKBC-WEKEX)*, Montreal, 2012.
13. Mathias Niepert, Jan Noessner, and Heiner Stuckenschmidt. Log-linear description logics. In Toby Walsh, editor, *IJCAI*, pages 2153–2158. IJCAI/AAAI, 2011.
14. Jan Noessner and Mathias Niepert. Elog: A probabilistic reasoner for owl el. In *Lecture Notes in Computer Science Web Reasoning and Rule Systems : 5th International Conference, RR 2011*, pages 281–286, Galway, Ireland, 2011. Springer.
15. Jan Noessner, Mathias Niepert, and Heiner Stuckenschmidt. Rockit: Rockit: Exploiting parallelism and symmetry for map inference in statistical relational models. In *Proceedings of the 27th Conference on Artificial Intelligence (AAAI)*, 2013.
16. Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging owl ontologies. In *Proceedings of the 14th international World Wide Web Conference*, page 633?640, Chiba, Japan, 2005.
17. Sebastian Riedel. Improving the accuracy and efficiency of map inference for markov logic. In David A. McAllester and Petri Myllymki, editors, *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pages 468–475, Helsinki, Finland, July 9-12 2008. AUAI Press.
18. Stefan Schlobach. Diagnosing terminologies. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05)*, page 670?675, 2005.
19. Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 355–362, Acapulco, Mexico, August 2003. Morgan Kaufmann.
20. Heiner Stuckenschmidt. Debugging owl ontologies - a reality check. In Raul Garcia-Castro, Asuncin Gmez-Prez, Charles J. Petrie, Emanuele Della Valle, Ulrich Kster, Michal Zaremba, and M. Omair Shafiq, editors, *Proceedings of the 6th International Workshop on Evaluation of Ontology-based Tools and the Semantic Web Service Challenge (EON-SWSC-2008)*, volume 359 of *CEUR Workshop Proceedings*, Tenerife, Spain, June 2008. CEUR-WS.org.
21. Renata Wassermann. An algorithm for belief revision. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*. Morgan Kaufmann, 2000.

Finding fault: Detecting issues in a versioned ontology

Maria Copeland, Rafael S. Gonçalves, Bijan Parsia, Uli Sattler and Robert Stevens

School of Computer Science, University of Manchester, Manchester, UK

Abstract. Understanding ontology evolution is becoming an active topic of interest to ontology engineers, e.g., we have large collaborative developed ontologies but, unlike software engineering, comparatively little is understood about the dynamics of historical changes, especially at a fine level of granularity. Only recently has there been a systematic analysis of changes across ontology versions, but still at a coarse-grained level. The National Cancer Institute (NCI) Thesaurus (NCIt) is a large, collaboratively-developed ontology, used for various Web and research-related purposes, e.g., as a medical research controlled vocabulary. The NCI has published ten years worth of monthly versions of the NCIt as Web Ontology Language (OWL) documents, and has also published reports on the content of, development methodology for, and applications of the NCIt. In this paper, we carry out a fine-grained analysis of the asserted axiom dynamics throughout the evolution of the NCIt from 2003 to 2012. From this, we are able to identify axiomatic editing patterns that suggest significant regression editing events in the development history of the NCIt.

1 Introduction

This paper is part of a series of analyses of the NCIt corpus [1,2], the earlier of which focus on changes to the asserted and inferred axioms. The current analysis extends previous work by tracing editing events at the individual axiom level, as opposed to the ontology level. That is, instead of analysing the total number of axioms added or removed between versions, we also track the appearance and disappearance of individual axioms across the corpus. As a result, we are able to positively identify a number of regressions (i.e., inadvertent introduction of an error) which occur over the last ten years of the development of the NCIt ontology, as well as a number of event sequences that, while not necessarily introducing errors, indicate issues with the editing process. We are able to do this analytically from the editing patterns alone.

2 Preliminaries

We assume that the reader is familiar with OWL 2 [3], at least from a modeller perspective. An ontology $\mathcal{O}$ is a set of axioms, containing logical and non-logical (e.g., annotation) axioms. The latter are analogous to comments in conventional programming languages, while the former describe entities (classes or individuals) and the relations between these entities via properties. The *signature* of an ontology $\mathcal{O}$ (the set of individuals, class and property names in $\mathcal{O}$) is denoted $\tilde{\mathcal{O}}$.

We use the standard notion of *entailment*, an axiom α entailed by an ontology $\mathcal{O}$ is denoted by $\mathcal{O} \models \alpha$. We look at entailments of the form $A \sqsubseteq B$ where A and B are class

names i.e., atomic subsumptions. This is the part of the type of entailment generated by the classification reasoning task, a standard reasoning task that forms the basis of the 'inferred subsumption hierarchy ' .

Finally, we use the notions of effectual and ineffectual changes as follows:

Definition 1. *Let $\mathcal{O}_i$ and $\mathcal{O}_{i+1}$ be two consecutive versions of an ontology $\mathcal{O}$.*
An axiom α is an addition (removal) *if $\alpha \in \mathcal{O}_{i+1} \backslash \mathcal{O}_i$ ($\alpha \notin \mathcal{O}_i \backslash \mathcal{O}_{i+1}$).*
An addition α is effectual *if $\mathcal{O}_i \not\models \alpha$ (written as $EffAdd(O_i, O_{i+1})$), and*
ineffectual *otherwise (written as $InEffAdd(O_i, O_{i+1})$).*
A removal α is effectual *if $\mathcal{O}_{i+1} \not\models \alpha$ (written as $EffRem(O_i, O_{i+1})$), and* ineffectual *otherwise (written as $InEffRem(O_i, O_{i+1})$) [1].*

3 Conceptual Foundations

Prior to the study of fault detection techniques, we establish a clear notion of the type of faults we are trying to isolate. In all cases, we define a *fault* as deviation from the required behaviour. In Software Engineering, software faults are commonly divided into functional and non-functional depending on whether the fault is in the required functional behaviour (e.g., whether the system is acting correctly in respect to its inputs, behaviour, and outputs) or whether the fault is in the expected service the system needs to provide (i.e., whether the (correct) behaviour is performed *well*). Functional and non-functional faults can be further subdivided based on the impact to the system and/or to the requirements specifications. For example, functional faults can be divided into fatal and non-fatal errors depending on whether the fault crashes the system. Generally, crashing behaviour is always a fatal fault, however it might be preferable to encounter a system crash instead of a non-fatal fault manifested in some other, harder to detect, manner. Faults that impact the requirements may be implicit, indeterminate (i.e., the behaviour might be *underspecified*), or shifting. A shifting specification can render previously correct behaviour faulty (or the reverse), as faults are defined as deviations from the "governing" specification. For convenience, we presume throughout this study that the specification is stable over the lifetime of the examined ontology, i.e., we expect the notation of 'acceptable model ' or 'acceptable entailment ' to be stable throughout the lifetime of the ontology.

We also restrict our attention to the *logical* behaviour of the ontology, and we approximate this by sets of desired entailments. This restriction might not reflect the full behaviour of an ontology in some application as 1) many entailments might be irrelevant to the application (e.g., non-atomic subsumptions for a terminologically oriented application) or 2) the application might be highly sensitive to other aspects of the ontology, including, but not limited to, annotations, axiom shape, and naming patterns. However, these other aspects are less standardised from application to application, so are rather more difficult to study externally to a given project. Furthermore, faults in the logical portion of an ontology both can be rather difficult to deal with and affect these other aspects. With this in mind, we define a logical bug as follows:

Definition 2. *An ontology $\mathcal{O}$ contains a (logical) bug if $\mathcal{O} \models \alpha$ and α is not a desired entailment or $O \not\models \alpha$ and α is a desired entailment.*

Of course, whether a (non)entailment is desired or not is not determinable by a reasoner — a reasoner can only confirm that some axiom is or is not an entailment. Generally, certain classes of (non)entailments are always regarded as errors. In analogy to crashing bugs in Software Engineering, in particular, the following are all standard errors:

1. $\mathcal{O}$ is inconsistent i.e., $\mathcal{O} \models \top \sqsubseteq \bot$
2. $A \in \widetilde{\mathcal{O}}$ is unsatisfiable in $\mathcal{O}$ i.e., $\mathcal{O} \models A \sqsubseteq \bot$
3. $A \in \widetilde{\mathcal{O}}$ is tautological in $\mathcal{O}$ i.e., $\mathcal{O} \models \top \sqsubseteq A$

In each of these cases, the "worthlessness" of the entailment is straightforward[1] and we will not justify it further here. That these entailments are bugs in and of themselves makes it easy to detect them, so the entire challenge of coping with such is in explaining and repairing them.

Of course, not all errors will be of these forms. For example, in most cases, the subsumption, $Tree \sqsubseteq Animal$ would be an undesired entailment. Detecting this requires domain knowledge, specifically, the denotation of `Tree` and `Animal`, the relation between them, and the intent of the ontology. If there is an explicit specification such e.g. a finite list of desired entailments, then checking for correctness of the ontology would be straightforward. Typically, however, the specification is implicit and, indeed, may be inchoate, only emerging via the ontology development process. Consequently, it would seem that automatic detection of such faults is impossible.

This is certainly true when considering a single version of an ontology. The case is different when multiple versions are compared. Crucially, if an entailment *fluctuates* between versions, that is, if it is the case that $\mathcal{O}_i \models \alpha$ and $\mathcal{O}_j \not\models \alpha$ where $i < j$, then we can conclude that *one* of those cases is erroneous. However, it is evident that $\mathcal{O}_i \not\models \alpha$ but $\mathcal{O}_j \models \alpha$ may not be as the fact that $\mathcal{O}_i \not\models \alpha$ as it might just indicate that the "functionality" has not been introduced yet. In what follows, we consider a sequence of $\mathcal{O}_i$, ... , $\mathcal{O}_m$ of ontologies, and use *i*, *j*, *k*, ... , as indices for these ontologies with *i*<*j*<*k*<... With this in mind, we can conservatively determine whether there are logical faults in the corpus using the following definition.

Definition 3. *Given two ontologies, $\mathcal{O}_i, \mathcal{O}_j$ where $i<j$, then the set of changes such that α is $\{EffAdd(O_i, O_{i+1}) \cap EffRem(O_j, O_{j+1})\}$is a* fault **indicating** set of changes *written as* FiSoC(i, j).

Note that if $\alpha \in FiSoC(i, j)$ either the entailment $\mathcal{O}_i \models \alpha$, thus $\alpha \in \mathcal{O}_i$, or the non-entailment $\mathcal{O}_i \not\models \alpha$ may be the bug in question and *FiSoC(i, j)* does not identify which is the bug. Instead the fault indicating set tells us that *one* of the changes introduces a bug. As mentioned earlier, the set shows the existence of a bug assuming a stable specification. Any subsequent findings of the same $\alpha \in FiSoC(i, j)$ is a fault indicate content regression. It is not surprising to find reoccurring content regressions due to the absence of content regression testing.

We can have a similar set of changes wherein the removal is *ineffectual* i.e., $\alpha \in \mathcal{O}_i$, $\alpha \notin \mathcal{O}_{i+1}$, but $\mathcal{O}_{i+1} \models \alpha$. Since the functionality of the ontology is not changed by

[1] There is, at least in the OWL community, reasonable consensus that these are all bugs in the sort of ontologies we build for the infrastructure we use.

an ineffectual removal, such a set does not indicate regression in the ontology. Indeed, such a set is consistent with a *refactoring* of the axiom, that is syntactic changes to the axiom that result in the axiom being strengthened or weakened based on the effectuallity of the change [1]. Of course, if the added axiom is the bug, then the ineffectual removal from $\mathcal{O}_i$ to $\mathcal{O}_{i+1}$ would be a failed attempt to remove the bug. Without access to developer intentions or other external information, we cannot distinguish between these two situations. However, we can conclude that an iterated pattern of ineffectual changes is problematic. That is, even if the set of changes $EffAdd(O_i, O_{i+1}) \cap InEffRem(O_j, O_{j+1})$ is a refactoring, a subsequent ineffectual addition, $InEffAdd(O_k, O_{k+1})$, would indicate a sort of thrashing. Meaning, if the original refactoring was correct, then "refactoring back" is a mistake (and if the "refactoring back" is correct, then the original refactoring is a mistake).

Definition 4. *Given two ontologies, $\mathcal{O}_i, \mathcal{O}_j$ where $i<j$, then any of the following sets of changes for α*

F1SSoC. $\{EffAdd(O_i, O_{i+1}) \cap InEffRem(O_j, O_{j+1})\}$
F2SSoC. $\{InEffAdd(O_i, O_{i+1}) \cap InEffRem(O_j, O_{j+1})\}$
F3SSoC. $\{InEffRem(O_i, O_{i+1}) \cap InEffAdd(O_j, O_{j+1})\}$

are fault **suggesting** set of changes *written as* FSSoC(i, j).

There is a large gap in the strength of the suggestiveness between sets of kind F1SSoC and the sets of kind F2SSoC and F3SSoC. Sets of kind F1SSoC can be completely benign, indicating only that additional information has been added to the axiom (e.g., that the axiom was strengthened), whereas there is no sensible scenario for the occurrence of sets of kind F2SSoC and F3SSoC. In all cases, much depends on whether the ineffectuality of the change is known to the ontology modeller. For instance, if a set of type F1SSoC(i, j) was an attempt to repair α, then α is a logical bug if α is an undesired entailment that was meant to have been repaired in O_j, then this repair failed.

All these suggestive sets may be embedded in larger sets. Consider the set where α is *(1)* $EffAdd(O_i, O_{i+1})$*, (2)* $InEffRem(O_j, O_{j+1})$*, (3)* $InEffAdd(O_k, O_{k+1})$*, (4)* $EffRem(O_l, O_{l+1})$. From this we have an indicative fault in the set <(1),(4)> and two suggestive faults in the sets, <(1),(2)> and <(2),(3)>. The latter two seem to be subsumed by the encompassing former. The analysis presented here does not, at this time, cover all paired possibilities. This is partly due to the fact that some are impossible on their own (e.g., two additions or two removals in a row) and partly due to the fact that some are subsumed by others.

Of course, as we noted, all these observations only hold if the requirements have been stable over the examined period. If requirements fluctuate over a set of changes, then the changes might just track the requirements and the ontology might never be in a pathological state.

4 Methods and Materials

The verification of the concepts and definitions proposed in Section 3 is carried out by conducting a detailed analysis of The National Cancer Institute Thesaurus (NCIt) ontology. The National Cancer Institute (NCI) is a U.S. government funded organisation for

the research of causes, treatment, and prevention of cancer [4]. The NCIt is an ontology written in the Web Ontology Language (OWL) which supports the development and maintenance of a controlled vocabulary about cancer research. Reports on the collaboration process between the NCIt and its contributors have been published in 2005 and 2009 (see [5,6,7]), which provide a view of the procedural practices adopted to support domain experts and users in the introduction of new concepts into the ontology. These publications together with the publicly available monthly releases and concept change logs are the basis for the corpus used in this study.

We gathered 105 versions of the NCIt (release 02.00 (October 2003) through to 12.08d (August 2012)) from the public website.[2] Two versions are unparseable using the OWL API [8], and were discarded, leaving 103 versions. The ontologies were parsed and individual axioms and terms were extracted and inserted into a MySQL v5.1.63 database. The database stores the following data for each NCIt release, $\mathcal{O}_i$ (where i is the version identifier):

1. Ontology $\mathcal{O}_i$: Each ontology's NCI identifier $\mathcal{O}_i$ is stored in a table "Ontology" with a generated integer identifier i.
2. Axioms $\alpha_j \in \mathcal{O}_i$: Each structurally distinct axiom α_j is stored in an "Axioms" table with identifier j, and a tuple (j, i) is stored in a table "Is In" (that is, axiom j is asserted in ontology i).
3. Classes $C_j \in \mathcal{O}_i$: Each class name C_j is stored in a table "Classes" with an identifier j, followed by the tuple (j, i) into table "Class In".
4. Usage of class C_j in $\mathcal{O}_i$: Each class C_j that is used (mentioned) in axiom $\alpha_k \in \mathcal{O}_i$ is stored in table "Used In" as a triple (j,k,i).
5. Effectual changes: Each added (removed) axiom $\alpha_j \in EffAdd(O_i, O_{i+1})$ ($\alpha_j \in EffRem(O_i, O_{i+1})$), with identifier j, is stored in table "Effectual Additions" ("Effectual Removals") as a tuple (j, $i+1$).
6. Ineffectual changes: Each added (removed) axiom $\alpha_j \in InEffAdd(O_i, O_{i+1})$ ($\alpha_j \in InEffRem(O_i, O_{i+1})$), with identifier j, is stored in table "Ineffectual Additions" ("Ineffectual Removals") as a tuple (j, i).

The data and SQL queries to produced this study are available online.[3]

All subsequent analysis are performed by means of SQL queries against this database to determine suitable test areas and fault detection analysis. For test area identification, we select test sets based on the outcome of 1) Frequency Distribution Analysis of the set of asserted axioms (i.e., in how many versions each axiom appears or follows), and 2) asserted axioms Consecutivity Analysis (whether an axiom's occurrence pattern has "gaps"). For fault detection, we conduct SQL driven data count analysis between the selected test cases and the Effectual and Ineffectual database tables to categorise logical bugs as FiSoCs or FSSoCs.

[2] `ftp://ftp1.nci.nih.gov/pub/cacore/EVS/NCI_Thesaurus/archive/`.

[3] `http://owl.cs.manchester.ac.uk/research/topics/ncit/regression-analysis/`

5 Results

5.1 Test Areas Selection

The test area selection for this study is determined by conducting analyses on axioms' frequency distribution and consecutivity evaluation. Frequency distribution analysis calculates the number of versions an axiom is present in the NCIt. From this sequence analysis we identify their consecutivity based on the type of occurrence in the corpus, such as: continual occurrence, interrupted occurrence, and single occurrence. The analysis of axioms with continual occurrence provides knowledge about the stability of the ontology, since it helps with the identification of axioms that, due to their consistent presence throughout the ontology's versions, can be associated with the 'core' of the represented knowledge. As described in Section 3, axioms' presence can be successfully correlated with FiSoCs or FSSoCs depending on the effectuality of their changes.

In the analysis, we found that the highest number of 20,520 asserted axioms correspond to frequency 11. This means that 20,520 axioms appear in the NCIt ontology for exactly 11 versions. Of these asserted axioms, 20,453 asserted axioms (99.67%), appear in 11 consecutive versions. The distribution of these axioms across the corpus is concentrated between version 6 to 16 with 19,384 asserted axioms (the majority of these additions took place in version 6 with 13,715 added axioms), between versions 1 to 52 with 593 asserted axioms, and 187 asserted axioms for the remaining versions. These numbers do not account for the 358 new asserted axioms added in version 93 that are still in the corpus for version 103 with 11 occurrences but have the potential of remaining in the corpus in the future versions.

The next highest frequency is 5 with 14,586 asserted axioms and 14,585 occurring consecutively. Only the axiom $\texttt{Extravasation} \sqsubseteq \texttt{BiologicalProcess}$ is present from version 20 to 23, it is removed in version 24 and re-enters in version 45 before being removed in version 46.

The next two rows in Table 1 show the results for frequency distribution 2 and 3 with 13,680 and 12,806 asserted axioms respectively. For frequency distribution 2, there are 10,506 asserted axioms with consecutive occurrence. Of these axioms, 445 entered the corpus in version 102 and remain in the corpus until version 103. The total number of axioms with non-consecutive occurrences is 3,174 asserted axioms. However, only 8 axioms are not included in the set of axioms that are part of the modification event taking place between versions 93 and 94. In this event 3,166 axioms with non-consecutive occurrences were added in version 93, removed (or possibly refactored) in version 94, and re-entered the ontology in version 103. This editing event is discussed in Section 6. Of the 12,806 asserted axioms with frequency distribution 3, 12,804 asserted axioms occur in consecutive versions (99.98%) and 644 asserted axioms are present in the last studied version of the corpus.

Our results show that three high frequency distributions are observed in the top ten distributions with axioms occurring in 87, 79 and 103 versions. There are 12,689 asserted axioms present in 87 versions with 99.86% of asserted axioms occurring consecutively. From these axioms, 12,669 asserted axioms appear in the last version of the ontology with 12,651 asserted axioms added in version 17 and remaining consecutively until version 103. For frequency distribution 79, there exist 10,910 asserted axioms

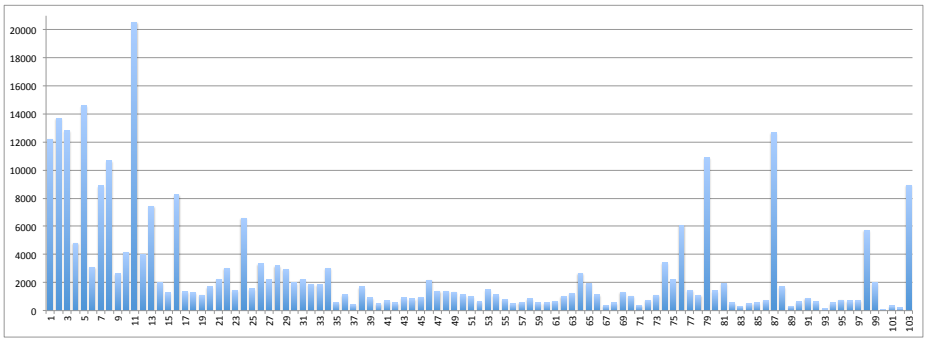

Fig. 1. Distribution of asserted axioms based on the number of versions they are present in (*x*-axis: frequency, *y*-axis: number of asserted axioms).

that appear in 79 versions with 10,866 still present in version 103. From these 10,866 asserted axioms, 10,861 asserted axioms were added in version 25 and remain until version 103 consecutively. Finally, there are 8,933 asserted axioms that appear in 103 versions of the NCIt. This means that 8,933 axioms were added in the first studied version of the NCIt and remain until the last studied version. That is, of the 132,784 asserted axioms present in version 103, 6.73% of the axioms were present from version 1. From this information it can be inferred that 6.73% of the asserted axioms population found in the last version of the NCIt represent a stable backbone of asserted axioms present in all versions of the NCIt.

Frequency	Axiom Count	Occurring in Version 103	Consecutive Occurrence	Non-consecutive Occurrence
11	20,520	358	99.67%	0.33%
5	14,586	831	99.99%	0.01%
2	13,680	445	76.80%	23.20%
3	12,806	664	99.98%	0.02%
87	12,689	12,669	99.86%	0.14%
1	12,219	47 in v102 and 2,084 in v103	–	–
79	10,910	10,866	99.93%	0.07%
8	10,662	599	99.93%	0.07%
103	8,933	8,933	100.00%	0.00%

Table 1. Frequency distribution trends.

As seen in Table 1, 12,219 asserted axioms occur in only 1 version of the NCIt. Of these asserted axioms, 2,084 axioms appear in version 103 and may remain in future versions. When taking this fact into account, we observe that a total of 10,135 asserted axioms with single occurrences are present in the remaining 102 versions. From the 103 studied versions, 98 versions have asserted axioms that only appear in those versions; and versions 45, 54, 88, 92, and 100 do not. A detailed representation of this distribution across the NCIt corpus demonstrates that the first three years of the studied NCIt versions show the highest rate of single occurrences in the corpus with three identifiable

high periods of single occurrences around version 1 to 5, versions 16 to 18, and versions 21 to 25.

5.2 Fault Detection Analysis

In this study, we limit Fault Detection Analysis to the finite set of asserted axioms with non-consecutive occurrence for the top ten frequencies identified in the previous section. It is important to note at this point that this study does not examine the set of all *FiSoC* and *FSSoC* for the collected versions of NCIt. Instead we focus our attention on the identified 53 asserted axioms that occur in non-consecutive versions for the top ten distributions, excluding all axioms that were part of the renaming events identified between versions 91 to 103 of the NCIt. Of these 53 examined axioms, 32 asserted axioms have logical bugs of type *FiSoC*. Further examination of the change sets of these *FiSoCs* indicate that 27 axioms conform directly with Definition 3 because all of their additions and removals are effectual; that is, the set of changes is $(EffAdd(O_i, O_{i+1}) \cap EffRem(O_j, O_{j+1}))$. The remaining 5 axioms have change sets of type $(EffAdd(O_i, O_{i+1}) \cap InEffRem(O_j, O_{j+1}) \cap EffRem(O_k, O_{k+1}))$. Although in this set there is an ineffectual removal prior to the effectual removal, from this change set we may conclude that ineffectual removal is "fixed" when the effectual removal takes place.

We also identified the asserted axiom `Benign_Peritoneal_Neoplasm` $\sqsubseteq$ `Disease_Has_Primary_Anatomic_Site only Peritoneum` with *axiom_id* 159025 as a logical bug of type *FiSoC* for the first removal $(EffAdd(O_{20}, O_{21}) \cap EffRem(O_{21}, O_{22}))$, and a second logical bug type *FSSoC* for the second removal $(EffAdd(O_{26}, O_{27}) \cap InEffRem(O_{28}, O_{29}))$. The presence of both logical bugs, *FiSoC* and *FSSoC*, in this axiom suggests that the reintroduction of the axiom to the ontology in version 27 after being removed in version 22 may correspond to content regression, and the second ineffectual removal in version 29 to refactoring.

The remaining 21 asserted axioms have logical bugs of type *FSSoC* . Seventeen of these axioms conform with *F1SSoC* set, thus suggesting possible refactoring. To confirm these refactoring events, additional axiomatic difference analysis needs to be carried out on these axioms, as suggested in [9]. Four axioms (*axiom_ids* 110594, 153578, 157661, and 127241) have the change sets identified for *F2SSoC*. Two of these axioms (*axiom_ids* 157661, and 127241) suggest refactoring for the first change set (the set is of type *F1SSoC*), and are later re-introduced in the ontology with logical bugs of type *FiSoC*.

As mentioned earlier, the analysis conducted in this section excludes fault detection for the set of axioms affected by the renaming event that took place between versions 91 and 103. We provide more information about this renaming event and the impact to our results in Section 6. However, it is important to mentioned that our analysis is sensitive to cosmetic changes to axioms, e.g., axiom renaming, and does not treat them as logical bugs due to the superfluous nature of these changes.

Frequency Rate	Axiom ID	Versions for <Eff. Add., Eff. Re.>	Versions for <Eff. Add.>	Versions for <Eff. Add., Ineff. Re., Eff. Re.>	Versions for <Ineff. Add.>	First NCIt Version	Last NCIt Version
11	57506	<4,5>, <7,17>				4	16
	58364	<4,5>, <7,17>				4	16
	103206			<7,17,26>		7	25
	105069			<7,17,26>		7	25
	210295	<40,47>, <51,55>				40	54
2	49544	<2,3>, <4,5>				2	4
	50602	<2,3>, <4,5>				2	4
	50858	<2,3>, <18,19>				2	18
	120551	<12,13>, <16,17>				12	16
	172613	<25,26>, <62,63>				25	62
	172917	<25,26>, <62,63>				25	62
3	159025	<21,22>				21	28
	257839	<83,84>, <93,94>	<103>			83	103
87	30433	<1,12>, <14,75>	<89>			1	103
	39267	<1,2>	<18>			1	103
	68617	<5,6>	<18>			1	103
	118516	<12,74>	<79>			12	103
	119326	<12,74>	<79>			12	103
	121919	<13,47>	<51>			13	103
	122832	<13,47>	<51>			13	103
79	6838			<1,17,86>	<23>	1	85
	8905	<1,6>	<30>			1	103
	44135			<1,17,86>	<23>	1	85
	125718	<15,19>	<29>			15	103
	125895	<15,19>	<29>			15	103
	162303	<23,93>, <94,103>				23	103
	162304	<23,34>	<34>			23	103
8	22465			<1,2,52>	<45>	1	51
	67505	<5,6>, <10, 17>				5	16
	238416	<72,79>	<103>			72	103
	238488	<72,79>	<103>			72	103
	262226	<87,93>, <94,96>				87	95

Table 2. Indicating fault in sequence of changes (Effectual Addition abbrv. to "Eff. Add.", Effectual Removal abbrv. to "Eff. Re.", Ineffectual Addition abbrv. to "Ineff. Add.", and Ineffectual Removal abbrv. to "Ineff. Re.").

6 Discussion

In general, the historical analysis of the NCIt, as recorded in their monthly releases from 2003 to 2012, show that the ontology is consistently active and the evolution management process in place for NCIt 's maintenance (as described in [10] and [6]) may be positive contributors to the overall steady growth of the NCIt ontology.

The growth of the ontology is mostly driven by the asserted ontology where high levels of editing activity took place in the first three years of the analysed population. The change dynamics observed in this period suggest a trial and error phase, where editing and modelling activities are taking place until reaching a level of stability, possibly related to reaching maturity, for the remainder of the observed versions.

Although the chronological analysis primarily points to the first three years as a phase of rapid change, a more in-depth study of the diachronic data set revealed that content regression takes place throughout all versions of the NCIt. A detailed study of the 'life of axioms' in the ontology from the Frequency Distribution Analysis shows that the evolution of the NCIt is marked by logical bugs of either *FiSoC* and/or *FSSoC* types.

Frequency Rate	Axiom ID	Versions for <Eff. Add., Ineff. Re.>	Versions for <Ineff. Add., Ineff. Re.>	Versions for <Ineff. Add.>	Versions for <Ineff. Add., Eff. Re.>	First NCIt Version	Last NCIt Version	Refactoring
11	110594		<10, 20>, <31, 32>			10	31	
	215592	<50, 55>		<98>		50	103	Refactoring
	215897	<50, 55>		<98>		50	103	Refactoring
5	157661	<20, 24>	<45, 46>			20	45	
2	99659	<6, 7>			<16, 17>	6	16	Refactoring
	127241	<16, 17>	<21, 22>			16	21	
3	159025	<27, 29>				21	28	Refactoring
87	3241	<1, 7>		<23>		1	103	Refactoring
	12085	<1, 17>		<33>		1	103	Refactoring
	106537	<9, 17>		<25>		9	103	Refactoring
	106569	<9, 17>		<25>		9	103	Refactoring
	106878	<9, 17>		<25>		9	103	Refactoring
	107407	<9, 17>		<25>		9	103	Refactoring
	107860	<9, 17>		<25>		9	103	Refactoring
	107952	<9, 17>		<25>		9	103	Refactoring
	108468	<9, 17>		<25>		9	103	Refactoring
	111380	<10, 17>		<24>		10	103	Refactoring
	114579	<10, 17>		<24>		10	103	Refactoring
79	42533	<1, 17>		<41>		1	103	Refactoring
8	153578		<17, 18>, <20, 27>			17	26	
	215709	<50, 53>		<99>		50	103	Refactoring

Table 3. Suggesting fault in sequence of changes (Effectual Addition abbrv. to "Eff. Add.", Effectual Removal abbrv. to "Eff. Re.", Ineffectual Addition abbrv. to "Ineff. Add.", and Ineffectual Removal abbrv. to "Ineff. Re.").

As a result, we found that asserted axioms with logical bugs enter the ontology in a version, are removed in a different version, and later re-entered the ontology unchanged. Only 6.73% of the asserted axioms in version 103 correspond to axioms that have been present unchanged from the first version analysed until this last version.

Our study revealed that most asserted axioms appear in two versions of the ontology. However, in this finding we identified 125,294 axioms are affected by the renaming event that took place between versions 93 and 94. In a preliminary study conducted for this paper, we found these asserted axioms first appear in version 93, are removed in version 94, and then re-enter the NCIt unchanged in version 103 . We have confirmed with the NCI that this editing event corresponds to the renaming of terms that took place in version 93, where every term name was replaced from its natural language name to its NCIt code. This renaming event also affects the set of asserted axioms with frequency distribution 11. The non-consecutive version occurrences for 1,186 axioms show that they first occur consecutively from versions 91 and 92, are removed in version 93, and then re-enter the ontology in version 94. These axioms remain consecutively until version 102 before they are removed again in version 103. The identification of this renaming event does not affect the information content dynamics of the ontology; however, it does affect the overall change dynamics. This renaming event is important to our analysis because it shows major editing periods are still part of the NCIt.

Taking into account these renaming events, the study found that the NCIt overall 'survival' rate for asserted axioms is 5 versions. Axioms with non-consecutive presence in the ontology are directly linked to logical bugs that either indicate content regressions or suggest axiom refactoring. Information content is not as permanent as the managerial and maintenance processes indicate, but logical bugs for unmodified axioms are more predominant than expected. The analysis conducted in this paper identifies specific sets

of axioms that are part of this group of regression cycles, and it is able to provide in detail the type of faulty editing patterns for these axioms and the location of these errors. We argue that the identification axioms with re-occurring logical bugs is a crucial step towards the identification of test cases and test areas that can be used systematically in Ontology Regression Testing.

7 Limitations

This study has taken under consideration the following limitations: *(i)* The NCIt evolution analysis and asserted axiom dynamics correspond to the publicly available OWL versions of the NCIt from release 02.00 (October 2003) to 12.08d (August 2012). Historical records of NCIt prior to OWL are not taken into consideration in this study. *(ii)* The presented results and analysis is limited in scope to the set of asserted axioms only. The inclusion of entailment analysis is only conducted in regards to the computation of logical differences to categorise the asserted axioms ' regression events into logical bugs of types *FiSoC* or *FSSoC*. *(iii)* Test area selection for the set of axioms with presence in non-consecutive versions is derived by selecting all axioms with non-consecutive presence based on their ranking in the high frequency analysis for all asserted axioms. The selected test area should be viewed as a snapshot of the whole population of axioms with non-consecutive presence, since the set of 53 analysed axioms correspond only to the top 10 high frequency distribution as described in Section 5.1. Analysis of the whole corpus is planned for future research. *(iv)* This study primarily corresponds to Functional Requirement Test Impact Analysis since it deals directly with the ontology. Non-functional Requirements are linked to entailment analysis such as subsumption hierarchy study, which is excluded in this work.

8 Conclusion

Large collaborative ontologies such as the NCIt need robust change analysis in conjunction with maintenance processes in order to continue to effectively support the ontology. The work presented in this paper shows that a detailed study of axioms with logical bugs need to be part of ontology evaluation and evolution analysis techniques due to its significant contribution to regression testing in ontologies. Although the study presented here is limited in that it is only evaluating unchanged asserted axioms, it still shows that a great portion of the editing efforts taking place in the NCIt is in the unmodified content. Regression analysis of this unmodified content can target specific changes in the modelling and representation approaches which can potential safe effort and increase productivity in the maintenance of the ontology.

Regression testing in Ontology Engineering is still a growing area of research, and the work presented here shows that a step towards achieving regression analysis in ontologies is by providing quantitative measurements of axiom change dynamics, identification of logical bugs, and the study of ontology evolutionary trends, all of which can be extracted efficiently by looking at versions of an ontology.

References

1. Gonçalves, R.S., Parsia, B., Sattler, U.: Analysing the evolution of the NCI thesaurus. In: Proc. of CBMS-11. (2011)
2. Gonçalves, R.S., Parsia, B., Sattler, U.: Analysing multiple versions of an ontology: A study of the NCI Thesaurus. In: Proc. of DL-11. (2011)
3. Cuenca Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P.F., Sattler, U.: OWL 2: The next step for OWL. J. of Web Semantics (2008)
4. de Coronado, S., Haber, M.W., Sioutos, N., Tuttle, M.S., Wright, L.W.: NCI Thesaurus: Using science-based terminology to integrate cancer research results. Studies in Health Technology and Informatics **107**(1) (2004)
5. Hartel, F.W., de Coronado, S., Dionne, R., Fragoso, G., Golbeck, J.: Modeling a description logic vocabulary for cancer research. J. of Biomedical Informatics **38**(2) (2005) 114–129
6. Thomas, N.: NCI Thesaurus - Apelon TDE Editing Procedures and Style Guide. National Cancer Institute. (2007)
7. Noy, N.F., de Coronado, S., Solbrig, H., Fragoso, G., Hartel, F.W., Musen, M.A.: Representing the NCI Thesaurus in OWL: Modeling tools help modeling languages. Applied Ontology **3**(3) (2008) 173–190
8. Horridge, M., Bechhofer, S.: The OWL API: A Java API for working with OWL 2 ontologies. In: Proc. of OWLED-09. (2009)
9. Gonçalves, R.S., Parsia, B., Sattler, U.: Categorising logical differences between OWL ontologies. In: Proc. of CIKM-11. (2011)
10. de Coronado, S., Wright, L.W., Fragoso, G., Haber, M.W., Hahn-Dantona, E.A., Hartel, F.W., Quan, S.L., Safran, T., Thomas, N., Whiteman, L.: The NCI Thesaurus quality assurance life cycle. Journal of Biomedical Informatics **42**(3) (2009)

Optique System: Towards Ontology and Mapping Management in OBDA Solutions

Peter Haase[2], Ian Horrocks[3], Dag Hovland[6], Thomas Hubauer[5], Ernesto Jimenez-Ruiz[3], Evgeny Kharlamov[3], Johan Klüwer[1] Christoph Pinkel[2], Riccardo Rosati[4], Valerio Santarelli[4], Ahmet Soylu[6], Dmitriy Zheleznyakov[3]

[1] Det Norske Veritas, Norway
[2] fluid Operations AG, Germany
[3] Oxford University, UK
[4] Sapienza University of Rome, Italy
[5] Siemens Corporate Technology, Germany
[6] University of Oslo, Norway

Abstract. The Optique project aims at providing an end-to-end solution for scalable Ontology-Based Data Access to Big Data integration, where end-users will formulate queries based on a familiar conceptualization of the underlying domain, that is, over an ontology. From user queries the Optique platform will automatically generate appropriate queries over the underlying integrated data, optimize and execute them. The key components in the Optique platform are the ontology and mappings that provide the relationships between the ontology and the underlying data. In this paper we discuss the problem of bootstrapping and maintenance of ontologies and mappings. The important challenge in both tasks is debugging errors in ontologies and mappings. We will present examples of different kinds of error, and give our preliminary view on their debugging.

1 Introduction

A typical problem that end-users face when dealing with Big Data is the data access problem, which arises due to the three dimensions (the so called "3V") of Big Data: *volume*, since massive amounts of data have been accumulated over the decades, *velocity*, since the amounts may be rapidly increasing, and *variety*, since the data are spread over a huge variety of formats and sources. In the context of Big Data, accessing the *relevant* information is an increasingly difficult problem. The Optique project[7] [5] aims at overcoming this problem.

The project is focused around two demanding use cases that provide it with motivation, guidance, and realistic evaluation settings. The first use case is provided by Siemens,[8] and encompasses several terabytes of temporal data coming from sensors, with a growth rate of about 30 gigabytes per day. Users need to query this data in combination with many gigabytes of other relational data that describe events. The second

[7] http://www.optique-project.eu
[8] http://www.siemens.com

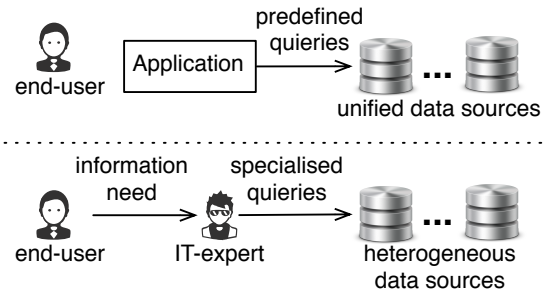

Fig. 1. Existing approaches to data access

use case is provided by Statoil[9], and concerns more than one petabyte of geological data. The data is stored in multiple databases which have different schemata, and the user has to manually combine information from many databases in order to get the results for a single query. In general, in the oil and gas industry, IT-experts spend 30–70% of their time gathering and assessing the quality of data [4]. This is clearly very expensive in terms of both time and money. The Optique project aims at solutions that reduce the cost of data access dramatically. More precisely, Optique aims at automating the process of going from an information requirement to the retrieval of the relevant data, and to reduce the time needed for this process from days to hours, or even to minutes. A bigger goal of the project is to provide a platform[10] with a generic architecture that can be easily adapted to any domain that requires scalable data access and efficient query execution for OBDA solutions.

The main bottleneck in the use cases discussed above is data access being limited to a restricted set of predefined queries (cf. Figure 1, top). Thus, if an end-user needs data that current applications cannot provide, the help of an IT-expert is required to translate the information need of the end-user to specialized queries and optimize them for efficient execution (cf. Figure 1, bottom). This process can take several days, and given the fact that in data-intensive industries engineers spend up to 80% of their time on data access problems [4] this incurs considerable cost.

The approach known as "Ontology-Based Data Access" (OBDA) [18,2] has the potential to address the data access problem by automating the translation process from the information needs of users to data queries (cf. Figure 2, left). The key idea is to use an ontology that presents to the user a conceptual model of the problem domain. The user formulates their information requirements (that is, queries) in terms of the ontology, and then receives the answers in the same intelligible form. These requests should be executed over the data automatically, without an IT-expert's intervention. To this end, a set of mappings is maintained which describes the relationships between the terms in the ontology and the corresponding data source fields.

In complex domains, a complete specification of the ontology and the mappings will typically be expensive to obtain, suggesting to start from partial specifications that are incrementally refined and expanded according to users' needs. Moreover, in some

[9] `http://www.statoil.com`

[10] Optique's solutions are going to be integrated via the Information Workbench platform [8].

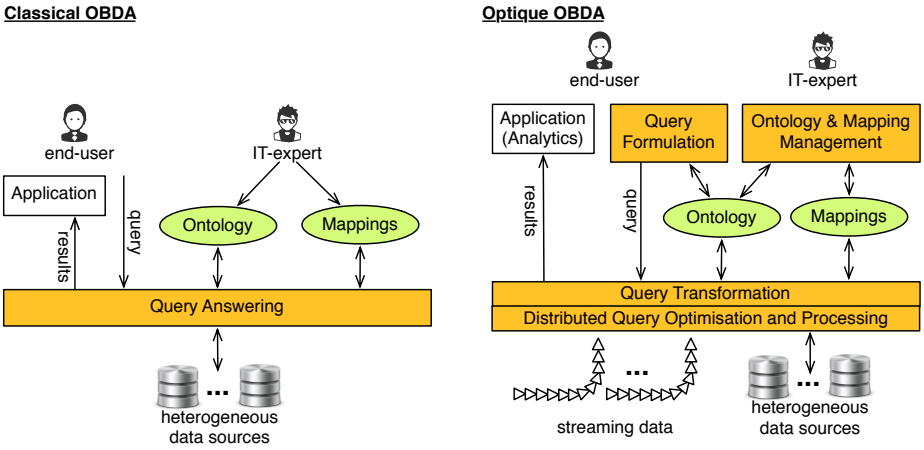

Fig. 2. Left: classical OBDA approach. Right: the Optique OBDA system

applications, changes in the ontology and/or in the schemata of the data sources (and thus in the mappings) are likely to happen. Thus, some means for bootstrapping and maintenance of ontology and mappings is required. The classical OBDA approaches fail to provide support for these tasks.

In the Optique project we aim at developing a next generation OBDA system (cf. Figure 2, right); more precisely, the project aims at a cost-effective approach that includes the development of tools and methodologies for semi-automatic bootstrapping of the system with a suitable initial ontology and mappings, and for updating them "on the fly" as needed by a given application. This means that, in our context, ontologies are dynamic entities that evolve *(i)* to incorporate new vocabulary required in users' queries, *(ii)* to accommodate new data sources, and *(iii)* to repair defects in ontologies and mappings. In all the cases, some way is needed to ensure that changes in the ontology and mappings are made in a coherent way. Due to this requirement, ontology *debugging* technologies will be a cornerstone of the system.

Besides ontology and mapping management, the Optique OBDA system will address a number of additional challenges, including: *(i)* user-friendly query formulation interface(s), *(ii)* processing and analytics over streaming data, *(iii)* automated query translation, and *(iv)* distributed query optimisation and execution in the Cloud. We will not, however, discuss these issues in this paper and refer the reader to [5] for details.

The remainder of the paper is organized as follows. First, we discuss different ontology defects that may arise and will need to be debugged (Section 2). Then we discuss how such defects can occur during the operation of the Optique OBDA system (Section 3), and how they will be dealt with.

2 OBDA Systems: Components and Defects

2.1 Components of OBDA Systems

An OBDA setting includes three central components: *data sources*, an *ontology*, and *mappings* (from data sources to the ontology).

Data sources. A data source consists of a data schema and a number of corresponding data instances. A typical example for a data source is a relational or semi-structured database.

Example 1. Consider the two data sources in Figure 3 (left): Source 1 (or S1 for short) contains a unary table about production wells with the schema $\mathbf{PWell}$ and says that the well 'w123' is a production well. Source 2 (or S2 for short) contains a unary table about exploration wells with the schema $\mathbf{EWell}$ and says that 'w123' is an exploration well. ■

Ontology. In the context of OBDA, it is usual to consider the ontology to be a Description Logic (DL) ontology [20] (or, equivalently, an OWL ontology). A DL ontology consists of a finite set of axioms that are usually in the form of set inclusions between two (possibly complexly defined) *concepts* that represent classes of objects. The ontology captures general knowledge about the domain of interest, such as generalizations, relational links, etc.

Example 2. Consider the ontology in Figure 3, left, (in the box). It describes (a part of) the oil production domain and consists of three concepts: *(i)* the concept $Well$ represents the class of *wellbores*,[11] *(ii)* the concept $PWell$ represents the class of *production wells*,[12] and *(iii)* the concept $EWell$ represents the class of *exploration wells*.[13] This ontology says that *production wells cannot be exploration wells* (denoted as $EWell \sqsubseteq \neg PWell$), *wells are exploration wells* ($Well \sqsubseteq EWell$), and *wells are production wells* ($Well \sqsubseteq PWell$). ■

Mappings. Mappings associate data from the data sources with concepts in the ontology.[14] A mapping m has the form

$$m : \mathbf{q}(\boldsymbol{x}) \rightsquigarrow N(\boldsymbol{x}),$$

where $\mathbf{q}$ is a query over the data sources, and N is an element of the ontological vocabulary, i.e., a concept or property name. Intuitively, $\mathbf{q}$ returns constants (resp., pairs of constants) from the data sources and *propagates* them into concept (resp., property) N (that is, *instantiates* N with them).

[11] A wellbore is any hole drilled for the purpose of exploration or extraction of natural resources, e.g., oil.

[12] *Production wells* are drilled primarily for producing oil or gas.

[13] *Exploration wells* are drilled purely for exploratory (information gathering) purposes in a new area.

[14] The mappings we are considering here are usually referred to as *global-as-view* mappings [13].

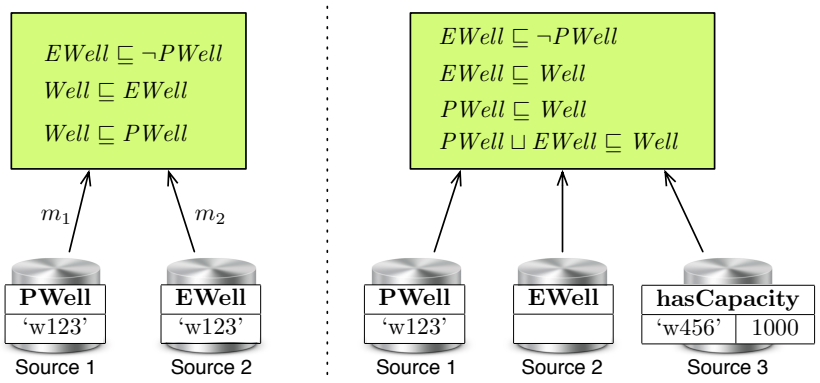

Fig. 3. Example of ontology and mapping defects. PWell stands for Production Well and EWell for Exploration Well.

Example 3. Consider Figure 3, left, again. The example includes two mappings, m_1 and m_2, which are depicted as arrows and defined as:

$$m_1 : \mathbf{PWell}(x) \rightsquigarrow PWell(x), \qquad m_2 : \mathbf{EWell}(x) \rightsquigarrow EWell(x).$$

Note that m_1 connects S1 with the concept $PWell$, while m_2 connects S2 with $EWell$: m_1 says that tuples from the table **PWell** are instances of the concept $PWell$, and m_2 analogously for $EWell$. I.e., m_1 instantiates $PWell$ and m_2 instantiates $EWell$. ■

Summing up Examples 1–3, we have the following OBDA setting depicted in Figure 3, left:

$$(\{S_1, S_2\}, \{EWell \sqsubseteq \neg PWell,\ Well \sqsubseteq EWell,\ Well \sqsubseteq PWell\}, \{m_1, m_2\}).$$

Another OBDA setting is illustrated in Figure 3, right, and we will comment on it in the following section.

In what follows, we present a number of defects that can occur in an OBDA setting and that may require debugging. We observe that several such defects have been individually pointed out and studied in literature (see, for instance, [17,19,16,11]). The Optique Project aims at facing these issues as a whole, and at individuating practical solutions for addressing them.

2.2 Logical and Modeling Defects

We distinguish two types of defects, namely logical and modeling.

The three types of logical defects that are usually discussed in the literature (see, for example, [22,15,21]) are *inconsistency*, *unsatisfiability*, and *incoherency*. Traditionally, these notions are applied to ontologies, while in the OBDA scenarios, as we will illustrate below, they may involve all three components of OBDA settings, that is, data sources, ontologies, and mappings.

Inconsistency of OBDA settings. An OBDA setting is *inconsistent* if it contains contradictory facts, so that there is no model of the ontology that is consistent with the data and the mappings.

Example 4. The OBDA setting in Figure 3, left, is inconsistent. Indeed, the well 'w123' is an instance of both concepts $PWell$ and $EWell$, and as the ontology asserts that $PWell$ and $EWell$ are disjoint, this makes the OBDA setting inconsistent. ■

The inconsistency in Example 4 could have been caused by a mistake in one of the data sources: either Source 1 or Source 2 provides wrong information about the well 'w123'. Perhaps, this well was an exploration one at the beginning, but then became a production one, but the information in Source 2 has not been brought up to date. A possible solution would be to update the offending data source. In Figure 3, right, there is an updated Source 2.

Unsatisfiability and Incoherency of Ontologies. This two types of defects are tightly related. A concept in an ontology is *unsatisfiable* if it cannot be instantiated without causing inconsistency, and an ontology is *incoherent* if an unsatisfiable concept occurs in it.

Example 5. Continuing with the OBDA setting in Figure 3, left, consider the concept $Well$. Observe that if we instantiate the concept with some object, say, 'w234', then it will turn out that 'w234' is both exploration and production well, which is inconsistent. Hence, the concept $Well$ is unsatisfiable and the ontology is incoherent. ■

Ontology incoherence is invariably indicative of an ontology design error. In Example 5, the two axioms stating that *wells are exploration wells* and *wells are productive wells* are in conflict with both our intuition and with the axiom stating that $PWell$ and $EWell$ are disjoint. Repairing this error will require one or more of these axioms to be modified or deleted. A possible repair plan is to replace the two counterintuitive axioms with axioms that make intuitive sense: $EWell \sqsubseteq Well$ (*exploration wells are wells*), and $PWell \sqsubseteq Well$ (*productive wells are wells*). Figure 3, right, contains a repaired version of the ontology.

Empty Mappings in OBDA settings. A mapping of an OBDA setting is *empty* if it does not propagate any individuals (resp., pairs of individuals) into any concept (resp., property) in the ontology.

Example 6. Consider the OBDA setting in Figure 3, right. Besides the mappings m_1 and m_2 from Example 3, it includes the following mapping:

$$m_3 : \mathbf{PWell}(x), \mathbf{hasCapacity}(x, y) \rightsquigarrow PWell(x).$$

that describes how to populate the concept $PWell$ by joining tuples from tables $\mathbf{PWell}$ and $\mathbf{hasCapacity}$ on the well's ID and projecting out the capacity value. Observe that the mapping m_3 is empty in this setting: when applying the mapping to the data

sources, no object will be propagated into the concept *PWell*, since there is no x such that it appears in both table **PWell** and table **hasCapacity**.

Now consider the following mappings:

$$m_4 : \mathbf{q}_1(x) \rightsquigarrow PWell(x);$$
$$m_5 : \mathbf{q}_1(x) \rightsquigarrow EWell(x);$$

Since the ontology states that $PWell \sqsubseteq \neg EWell$, it follows that the query q_1 has to be empty, i.e., its evaluation over the data sources must return the empty tuple. ■

The defect with the mapping m_3 in Example 6 could be caused by *(i)* a mapping-design error, that is, tables $\mathbf{PWell}$ and $\mathbf{hasCapacity}$ are incorrectly related in m_3, or *(ii)* incompleteness of data sources. In the former case, m_3 should be deleted or repaired, while in the later case the problem should be solved at the data source level. Moreover, mappings m_4 and m_5 show that combining the knowledge about the mapping and the ontology is a key aspect towards the formal analysis of the OBDA specification.

Modeling defects are less intuitive than logical ones. A typical modeling defect is *redundancy* [7].

Redundancy. We distinguish three types of redundancy: *redundant axioms*, *concepts*, and *mappings*. Intuitively, an axiom, concept, or mapping is redundant in an OBDA setting if the deletion of it from the setting results in a logically *equivalent* setting.

Example 7. Recall the OBDA system in Figure 3, right. To observe the phenomenon of redundant axioms, note that the axiom $\alpha = PWell \sqcup EWell \sqsubseteq Well$ (*both production and exploration wells are wells*) would be redundant in this setting. Indeed, the pair of axioms $PWell \sqsubseteq Well$ and $EWell \sqsubseteq Well$ already state the same thing, i.e., they are logically equivalent to α, and so adding α would be vacuous.

To observe the phenomenon of redundant concepts, assume that the ontology implies that two concepts, say $PWell$ and $ExWell$ (standing for Exploited Well) are equivalent, i.e., $PWell \sqsubseteq ExWell$ and $ExWell \sqsubseteq PWell$. Then, these two concepts are synonymous, and we may prefer to remove one of them from the ontology.[15] Even if the ontology does not imply the logical equivalence of $PWell$ and $ExWell$, but the mappings for these concepts are the same, then the two concepts may be de facto synonymous.

To observe the phenomenon of redundant mappings, consider the mapping m_3 (Example 6) and the following mapping m_5:

$$m_6 : \mathbf{PWell}(x) \rightsquigarrow PWell(x).$$

Note that, in the presence of m_6, m_3 becomes redundant. Indeed, m_3 instantiates the concept $PWell$ with *some* objects from the table $\mathbf{PWell}$, while m_6 instantiates $PWell$ with *all* the objects found in $\mathbf{PWell}$. Thus, m_3 can be harmlessly dropped. ■

[15] This is not always the case as synonyms may capture differences in vocabulary usage amongst different user groups.

To conclude this section, we would like to note that it is not trivial to understand whether a modeling defect is actually a defect and hence requires debugging. For example, as noted above, redundancy can be intentionally introduced in an ontology by an ontology engineer. Thus, the necessity of debugging such errors depends on the application, and should be decided on a case-by-case basis.

3 Supporting the Life Cycle of OBDA Systems

Essential functionalities, required to support the life cycle of an OBDA system, are:

- Detection of defects,
- OBDA debugging,
- Ontology and mapping bootstrapping,
- OBDA evolution,
- OBDA transformation.

We will now discuss these functionalities in more detail.

An OBDA system should be able to analyze itself w.r.t. both logical and modeling defects as presented above. Thus, the system should be equipped with an OBDA *analyser*: a routine that takes an OBDA setting as input, and returns a set of defects as output. Based on the result of such analyses, the system should be able to debug itself. Since, as we discussed in the previous section, there is no universal way to debug an OBDA system, it is natural for the debugging to be semi-automatic. Thus, the system should be equipped with an OBDA *debugger*: a routine that takes an OBDA setting and its defects as input, and returns a debugged version of the setting, that is free from the input defects. For examples of tools that perform these tasks, we refer the reader to [16,12].

Clearly, OBDA systems crucially depend on the existence of suitable ontologies and mappings. Developing them from scratch is likely to be expensive and a practical OBDA system should support a (semi-) automatic bootstrapping of an initial ontology and set of mappings. Thus, an OBDA system should be equipped with an OBDA *bootstrapper*: a routine that takes a set of database schemata and possibly instances over these schemata as input, and returns an ontology and mappings connecting it to the input schemata.

As was discussed above, OBDA systems are not static objects and are subjects of frequent changes, that is, evolution. Natural types of evolution are:

- **Adding/deleting a new concept together with a mapping.** It might be the case that the ontology lacks some concepts, or some concept should be dropped from the ontology, e.g., when it is synonymous with another concept. For example, consider the ontology in Figure 3, right. Assume that a new concept, e.g., $OilPr$ (*Oil Producer*), is needed. Then one might add this concept to the ontology and create a mapping that instantiates the concept.
- **Adding/deleting an ontological axiom.** An ontology may be missing relations between concepts. For example, one may want to assert that the concept $OilPr$ is a subconcept of $Well$. This can be done by adding the axiom $OilPr \sqsubseteq Well$ to the ontology.

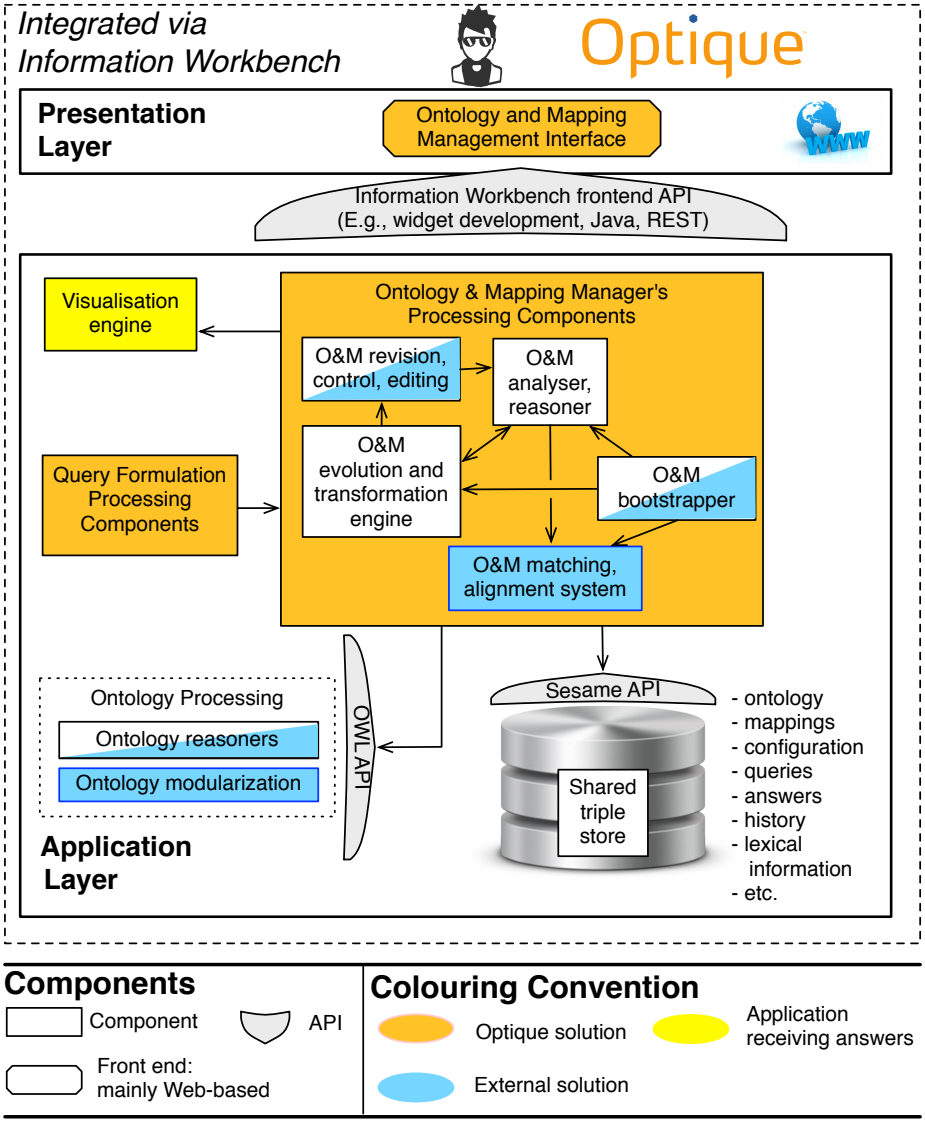

Fig. 4. Ontology and Mapping Management component of the Optique OBDA system

– **Mapping modification.** Mappings can be added, changed, or deleted. For example, if a new data source is added, one can create mappings from the new data source to some existing concepts. Another scenario for mapping modifications is optimization, that is, one may adjust mappings based on the constraints in the data sources in order to improve the performance of query processing.

A task related to both bootstrapping and evolution is ontology transformation. For example, one may want to reuse an existing third party ontology or mappings instead of (or combined with) bootstrapping them from the data sources. It is possible that the existing ontology or mappings are in a language that the system at hand does not support. Thus, one will have to transform them into the supported language. This may involve changes in the syntax and/or expressivity of the ontology.

Another reason for a transformation is optimization. One may want to improve the overall performance of an OBDA system by restricting the expressiveness of the ontology language, e.g., by moving from an OWL 2 ontology O to an OWL 2 QL ontology O'. This would in general require "approximation" of O using the weaker OWL 2 QL language [14,1].

To connect the five functionalities above, we observe that bootstrapping, evolution, and transformation functionalities naturally introduce errors in OBDA settings, while the detection functionality can detect the errors, and the debugging one can repair them.

In the following section, we present the Optique OBDA system, which will provide all of the life cycle supporting functionalities described above.

4 Optique OBDA Ontology and Mapping Manager

In Figure 4, we present the *Ontology and Mapping Management* component (the O&M manager) of the Optique OBDA system. The O&M manager has a Web interface at the presentation layer. Functionalities of the O&M manager are intended for IT-experts rather than end-users. The manager includes five subcomponents:

- The *Bootstrapper* extracts an initial ontology and mappings from data sources, as discussed above. In Section 4.1, we will present our initial ideas on how to implement the bootstrapper.
- The *Matching and alignment system* performs ontology alignment.
- The *Analyser and reasoner* checks ontologies for defects.
- The *Evolution and transformation engine* performs debugging on defects found by the analyser.
- The *Revision, control and editing system* supports versioning (which can be based on, e.g., ContentCVS [10]) and editorial processes for both ontologies and mappings. It can also act as a hub, coordinating interoperation between the analyser and evolution engine.

Also, the O&M manager interacts with other components of the Optique OBDA system. In particular,

- It accesses the *Shared triple store*, where the ontology and mappings are physically stored. It can both read the ontology and mappings and update them when needed.
- The analyser, alignment system, and evolution engine have access to reasoning capabilities, e.g., external ontology reasoners, ontology modularization engines, etc.
- The *Query Formulation Component* can call the O&M manager whenever a user decides to add a new concept or axiom. We refer to this as query-driven ontology construction.
- Finally, the O&M manager is connected to a *Visualisation engine*.

4.1 Directions

In the development of the Optique OBDA system, we plan to exploit existing techniques from ontology evolution, e.g. [3,6], ontology modularisation, e.g. [23], and develop our own novel techniques. For example, a possible bootstrapping technique could be the following three step procedure:

- Step 1: Bootstrap direct[16] and R2RML[17] mappings from relational schemata of the data sources. These mappings naturally give ontological vocabularies over the schemata.
- Step 2: Construct a simple ontology over these vocabularies by extracting ontological axioms from the integrity constraints of the schemata.
- Step 3: Align the simple ontologies and the vocabularies with a state of the art ontology using an existing system, e.g. LogMap [9].

5 Conclusions

We have presented a selection of possible logical and modeling errors in OBDA systems and the main challenges to be faced in supporting the life-cycle of OBDA systems. Current approaches and methods only partially address the issues related to the construction, maintenance, and transformation of an OBDA specification. Although the EU project Optique is still at an early stage, we aim to turn our preliminary ideas into novel solutions in the very near future, and to evaluate their effectiveness in our industry use-cases. This will provide us with invaluable feedback to inform ongoing research and development of enhanced ontology and mapping management components.

Acknowledgements

The research presented in this paper was financed by the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, the Optique project. Horrocks, Jiménez-Ruiz, Kharlamov, and Zheleznyakov were also partially supported by the EPSRC projects ExODA and Score!

References

1. Botoeva, E., Calvanese, D., Rodriguez-Muro, M.: Expressive approximations in DL-Lite ontologies. Proc. of AIMSA 2010 pp. 21–31 (2010)
2. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO System for Ontology-Based Data Access. Semantic Web 2(1), 43–53 (2011)
3. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Evolution of DL-Lite Knowledge Bases. In: International Semantic Web Conference (1). pp. 112–128 (2010)

[16] http://www.w3.org/TR/2011/WD-rdb-direct-mapping-20110324/
[17] http://www.w3.org/TR/r2rml/

4. Crompton, J.: Keynote talk at the W3C Workshop on Semantic Web in Oil & Gas Industry: Houston, TX, USA, 9–10 December (2008), available from http://www.w3.org/2008/12/ogws-slides/Crompton.pdf
5. Giese, M., Calvanese, D., Haase, P., Horrocks, I., Ioannidis, Y., Kllapi, H., Koubarakis, M., Lenzerini, M., Möller, R., Özep, O., Rodriguez Muro, M., Rosati, R., Schlatte, R., Schmidt, M., Soylu, A., Waaler, A.: Scalable End-user Access to Big Data. In: Rajendra Akerkar: Big Data Computing. Florida: Chapman and Hall/CRC. To appear. (2013)
6. Grau, B.C., Jiménez-Ruiz, E., Kharlamov, E., Zheleznyakov, D.: Ontology Evolution Under Semantic Constraints. In: KR 2012 (2012)
7. Grimm, S., Wissmann, J.: Elimination of Redundancy in Ontologies. In: ESWC (1). pp. 260–274 (2011)
8. Haase, P., Schmidt, M., Schwarte, A.: The Information Workbench as a Self-Service Platform for Linked Data Applications. In: COLD (2011)
9. Jiménez-Ruiz, E., Grau, B.C.: LogMap: Logic-Based and Scalable Ontology Matching. In: International Semantic Web Conference (1). pp. 273–288 (2011)
10. Jiménez-Ruiz, E., Grau, B.C., Horrocks, I., Llavori, R.B.: Supporting Concurrent Ontology Development: Framework, Algorithms and Tool. Data Knowl. Eng. 70(1), 146–164 (2011)
11. Keet, C.M., Alberts, R., Gerber, A., Chimamiwa, G.: Enhancing web portals with ontology-based data access: the case study of south africas accessibility portal for people with disabilities. In: Proc. of OWLED 2008. vol. 2008 (2008)
12. Lehmann, J., Bühmann, L.: ORE - A tool for repairing and enriching knowledge bases. In: Proc. of ISWC 2010. Springer (2010)
13. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: PODS. pp. 233–246 (2002)
14. Pan, J.Z., Thomas, E.: Approximating OWL-DL ontologies. p. 1434 (2007)
15. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging OWL Ontologies. In: WWW. pp. 633–640 (2005)
16. Poveda-Villalón, M., Suárez-Figueroa, M.C., Gómez-Pérez, A.: Validating ontologies with OOPS! In: Proc. of EKAW 2012, pp. 267–281. Springer (2012)
17. Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In: Proc. of EKAW 2004. pp. 63–81. Springer (2004)
18. Rodriguez-Muro, M., Calvanese, D.: High Performance Query Answering over DL-Lite Ontologies. In: KR (2012)
19. Roussey, C., Corcho, O., Vilches-Blázquez, L.M.: A catalogue of OWL ontology antipatterns. In: Proc. of K-CAP 2009. pp. 205–206. ACM (2009)
20. Sattler, U., Calvanese, D., Molitor, R.: Relationships with other Formalisms. In: Description Logic Handbook. pp. 137–177 (2003)
21. Shchekotykhin, K., Friedrich, G., Fleiss, P., Rodler, P.: Interactive Ontology Debugging: Two Query Strategies for Efficient Fault Localization. Web Semantics: Science, Services and Agents on the World Wide Web 12(0) (2012)
22. Stuckenschmidt, H.: Debugging OWL Ontologies - A Reality Check. In: EON (2008)
23. Vescovo, C.D., Parsia, B., Sattler, U.: Logical Relevance in Ontologies. In: Description Logics (2012)

Repairing missing is-a structure in ontologies is an abductive reasoning problem

Patrick Lambrix[1,2], Fang Wei-Kleiner[1], Zlatan Dragisic[1,2], Valentina Ivanova[1,2]

(1) Department of Computer and Information Science, (2) Swedish e-Science Research Centre
Linköping University, 581 83 Linköping, Sweden

Abstract. With the increased use of ontologies in semantically-enabled applications, the issue of debugging defects in ontologies has become increasingly important. These defects can lead to wrong or incomplete results for the applications. Debugging consists of the phases of detection and repairing. In this paper we focus on the repairing phase of a particular kind of defects, i.e., the missing relations in the is-a hierarchy. We show that this can be formalized as an abduction problem. Further, we define properties for the ontology, the set of is-a relations to repair and the domain expert, as well as preference criteria on solutions and discuss the influences of these properties and criteria on the existence of solutions for the abduction problem. We also discuss the consequences of our analyses of the repairing problem for the development and use of debugging systems.

1 Introduction

Developing ontologies is not an easy task, and often the resulting ontologies are not consistent or complete. Such ontologies, although often useful, also lead to problems when used in semantically-enabled applications. Wrong conclusions may be derived or valid conclusions may be missed. Defects in ontologies can take different forms (e.g., [16]). Syntactic defects are usually easy to find and to resolve. Defects regarding style include such things as unintended redundancy. More interesting and severe defects are the modeling defects which require domain knowledge to detect and resolve, and semantic defects such as unsatisfiable concepts and inconsistent ontologies. Debugging consists of two phases - detection and repair. Most work up to date has focused on debugging the semantic defects in an ontology (see related work in Section 5).

Modeling defects have mainly been discussed for taxonomies, i.e., from a knowledge representation point of view, a simple kind of ontologies. The focus has been on defects regarding the is-a structure (Section 5). In addition to its importance for the correct modeling of a domain, the structural information in ontologies is also important in semantically-enabled applications such as ontology-based search and annotation. In this paper we formalize the problem of repairing the is-a structure of ontologies.

There are different ways to detect missing is-a relations (Section 5). One way is inspection by domain experts. Another way is to use ontology learning techniques or patterns. When the ontology is part of a network of ontologies connected by mappings, missing is-a relations may be detected using logical derivation in the network. However, although there are many approaches to detect missing is-a relations, these approaches,

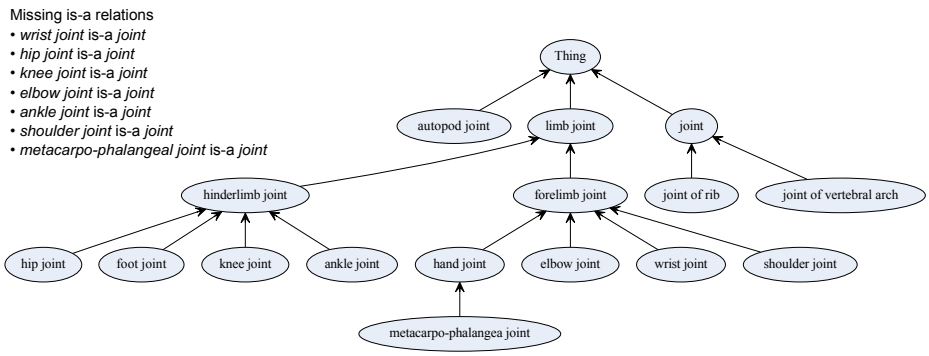

Fig. 1: A part of MA concerning the concept *joint*.

in general, do not detect *all* missing is-a relations. For instance, although the precision for the linguistic patterns approaches is high, their recall is usually very low.

In this paper we assume that the detection phase has been performed. We assume that we have obtained a set of missing is-a relations for a given ontology (validated or not) and focus on the repairing phase. In the ideal case where our set of missing is-a relations contains *all* missing is-a relations, the repairing phase is easy. We just add all missing is-a relations to the ontology and a reasoner can compute all logical consequences. However, when the set of missing is-a relations does not contain all missing is-a relations - and this is the common case - there are different ways to repair the ontology.

For instance, Figures 1 and 2 (T) show a small ontology representing a part of the Adult Mouse Anatomy (MA) ontology concerning joint, that is relevant for our discussion. M is a set of detected missing is-a relations. Adding these relations to the ontology will repair the missing is-a structure. However, there are other more interesting possibilities. For instance, adding limb-joint $\dot{\sqsubseteq}$ joint also repairs the missing is-a structure. Further, this is-a relation is correct according to the domain and constitutes a new is-a relation that was not derivable from the ontology and not originally detected by the detection algorithm.

The contributions of this paper are the following. First, in Section 2 we formalize the problem of repairing missing is-a structure as an abduction problem (extension of [18]) and introduce two decision problems - (i) do solutions exist, and (ii) if so, find a solution. We also define different properties for the ontology, the set of is-a relations to repair, and the domain expert and discuss the influences of these properties on the existence of solutions for the abduction problem. In general, when solutions exist, there may be many solutions. As not all solutions are equally interesting, in Section 3 we propose two preference criteria on the solutions as well as different ways to combine these. We also discuss the decision problems for the criteria and their preferences. Further, in Section 4 we discuss the consequences of our analyses for debugging in practice.

C = { autopod-joint, limb-joint, hinderlimb-joint, hip-joint, foot-joint, knee-joint, ankle-joint, forelimb-joint, hand-joint, elbow-joint, wrist-joint, shoulder-joint, metacarpo-phalangeal-joint, joint, joint-of-rib, joint-of-vertebral-arch }

T = { autopod-joint $\dot{\sqsubseteq}$ $\top$, limb-joint $\dot{\sqsubseteq}$ $\top$, hinderlimb-joint $\dot{\sqsubseteq}$ limb-joint , hip-joint $\dot{\sqsubseteq}$ hinderlimb-joint, foot-joint $\dot{\sqsubseteq}$ hinderlimb-joint, knee-joint $\dot{\sqsubseteq}$ hinderlimb-joint, ankle-joint $\dot{\sqsubseteq}$ hinderlimb-joint, forelimb-joint $\dot{\sqsubseteq}$ limb-joint, hand-joint $\dot{\sqsubseteq}$ forelimb-joint, elbow-joint $\dot{\sqsubseteq}$ forelimb-joint, wrist-joint $\dot{\sqsubseteq}$ forelimb-joint, shoulder-joint $\dot{\sqsubseteq}$ forelimb-joint, metacarpo-phalangeal-joint $\dot{\sqsubseteq}$ hand-joint, joint $\dot{\sqsubseteq}$ $\top$, joint-of-rib $\dot{\sqsubseteq}$ joint, joint-of-vertebral-arch $\dot{\sqsubseteq}$ joint }

M = { wrist-joint $\dot{\sqsubseteq}$ joint, hip-joint $\dot{\sqsubseteq}$ joint, knee-joint $\dot{\sqsubseteq}$ joint, elbow-joint $\dot{\sqsubseteq}$ joint, ankle-joint $\dot{\sqsubseteq}$ joint, shoulder-joint $\dot{\sqsubseteq}$ joint, metacarpo-phalangeal-joint $\dot{\sqsubseteq}$ joint }

H_1 = set of all is-a relations that are correct according to the domain
H_2 = $H_1 \setminus$ {autopod-joint $\dot{\sqsubseteq}$ limb-joint, limb-joint $\dot{\sqsubseteq}$ joint}
H_3 = $H_2 \cup$ {hinderlimb-joint $\dot{\sqsubseteq}$ joint-of-rib, forelimb-joint $\dot{\sqsubseteq}$ joint-of-vertebral-arch}
H_4 = { A $\dot{\sqsubseteq}$ B | A, B $\in C$ }

Let $\mathcal{P}_i$ = GTAP(T, C, H_i, M) for $1 < i < 4$

Fig. 2: Small example.

2 Abduction Framework

In the following we explain how the problem of finding possible ways to repair the missing is-a structure in an ontology is formalized as a generalized version of the TBox abduction problem (extension of [18]). We assume that our ontology is represented using a TBox T. The identified is-a relations to repair are then represented by a set M of atomic concept subsumptions. As discussed in Section 1, M usually does not contain *all* missing is-a relations. To repair the ontology, it should be extended with a set S of atomic concept subsumptions (repair) such that the extended ontology is consistent and the missing is-a relations are derivable from the extended ontology. However, the added atomic concept subsumptions should be correct according to the domain[1]. Therefore, we assume that a domain expert validates whether an atomic concept subsumption is correct and these validated to be correct atomic concept subsumptions are collected in a set H. We note that in practice H is not known beforehand, but acts as an oracle. It is then required that $S \subseteq H$. The following definition formalizes this.

Definition 1 (Generalized TBox Abduction) *Let T be a consistent TBox and C be a set of atomic concepts. Let $M = \{\ C_i \dot{\sqsubseteq} D_i \mid 1 \leq i \leq m\ \}$ be a set of TBox assertions where $C_i, D_i \in C$. Let $H = \{E_i \dot{\sqsubseteq} F_i \mid 1 \leq i \leq n\}$ where $E_i, F_i \in C$. A solution to the generalized TBox abduction problem (GTAP) (T, C, H, M) is any finite set $S \subseteq H$, such that $T \cup S$ is consistent and $T \cup S \models M$. The set of all such solutions is denoted as $\mathcal{S}(T, C, H, M)$.*

Moreover, we are interested in two problems which are useful in practice. The first problem is the so called existence problem. That is, the decision problem of whether $\mathcal{S}(T, C, H, M) \neq \emptyset$. Clearly, with a concrete debugging task the existence problem should be answered at the beginning. If the answer to the existence problem is positive,

[1] In the remainder of this paper when we say that concept subsumptions or is-a relations are *correct*, we mean correct according to the domain.

we are interested in finding *a* solution[2]. This is normally a realistic goal in practice, since the number of all solutions could be considerably big.

Next, we discuss different properties of T, H and M and how these properties and their combinations affect the existence and type of solutions. In this discussion we make the assumption that the domain is consistent.

The GTAP definition requires T to be consistent. If this would not be the case, it would mean that the original ontology is not consistent. In this case approaches for debugging semantic defects could be used to obtain a consistent ontology. We also note that if T is not consistent then there are no solutions satisfying the definition (as $T \cup S$ would be inconsistent). However, even if T is consistent, it is possible that T contains relations which are not correct. It would mean that the developers introduced a modeling defect. Therefore, we indentify two cases for T - all the is-a relations in T are correct ('T correct' in Table 1), or not ('T not correct' in Table 1).

For M there are 2 cases. In the first case we assume that all is-a relations in M are correct, and thus they are really missing is-a relations ('Missing' in Table 1). In the second case M may contain missing as well as wrong is-a relations ('Missing + Wrong' in Table 1). This is a common case when possible missing is-a relations are generated by detection algorithms (e.g., using patterns or ontology learning methods) and not validated by a domain expert. It may also occur when M is generated by domain experts (e.g., using inspection) - as it is an error-prone task, the experts may make mistakes.

For H we identified the following interesting cases. In the first case ('Complete Knowledge' in Table 1) H contains all correct is-a relations and no others. In this case we are sure that if an is-a relation belongs to H, it is correct and if not, it is not correct. This case represents the ideal situation of an all-knowing domain expert. In the second case ('Partial-Correct' in Table 1) H contains only correct is-a relations, but not necessarily all. This case represents a domain expert who knows a part of the domain well. If the domain expert validates an is-a relation as correct, it is correct. Otherwise, the is-a relation is wrong or the domain expert does not know. An approximation of this case is when using several domain experts and a skeptical approach. We only consider an is-a relation correct if all domain experts validate it as correct. In the third case ('Wrong' in Table 1) H may contain relations that are not correct. In this case, the domain expert can make mistakes regarding the validation of is-a relations. Some wrong is-a relations may be validated as correct. This is a common case as exemplified by the use case in [12]. The fourth and fifth cases represent situations where there is no domain expert. In the fourth case all possible is-a relations are validated as correct and thus $H = \{E_i \dot{\sqsubseteq} F_i \mid E_i, F_i \in C\}$ ('No Expert' in Table 1). In the fifth case (not in in Table 1) no is-a relation is validated as correct and thus $H = \emptyset$. For the fifth case there can be only 1 solution, i.e., S = $\emptyset$ and this only in the case where $T \models M$ (and thus the is-a relations in M were not actually missing). We have the following relations between the different cases. Let H_c, H_{pc}, H_w, H_{no} be sets corresponding to the cases 1-4, respectively and related to the same domain. Then $H_{pc} \subset H_c \subset H_{no}$ and $H_w \subset H_{no}$. Therefore, we also have that $\mathcal{S}(T, C, H_{pc}, M) \subset \mathcal{S}(T, C, H_c, M) \subset \mathcal{S}(T, C, H_{no}, M)$ and $\mathcal{S}(T, C, H_w, M) \subset \mathcal{S}(T, C, H_{no}, M)$. In our example in Figure 2 H_1, H_2, H_3 and H_4 are examples of H_c, H_{pc}, H_w and H_{no}, respectively.

[2] Often regarding various preference criteria, see Section 3.

M / H	Missing: T correct	Missing: T not correct
Complete Knowledge	$M \subseteq H$ M is solution All solutions are correct	$M \subseteq H$ No solution if $T \cup M$ inconsistent M is solution iff $T \cup M$ consistent All solutions are correct
Partial-Correct	$M \subseteq H$ or $M \not\subseteq H$ No solution if $M \not\subseteq H \wedge T \cup H \not\models M$ if $M \subseteq H$ then M is a solution if $M \not\subseteq H \wedge T \cup H \models M$ then H is a solution All solutions are correct	$M \subseteq H$ or $M \not\subseteq H$ No solution if $T \cup M$ inconsistent No solution if $M \not\subseteq H \wedge T \cup H \not\models M$ No solution if $\forall S : S \neq \emptyset \wedge S \subseteq H \rightarrow T \cup S$ inconsistent if $T \cup M$ consistent $\wedge$ $M \subseteq H$ then M is a solution if $T \cup H$ consistent $\wedge$ $M \not\subseteq H \wedge T \cup H \models M$ then H is a solution All solutions are correct
Wrong	$M \subseteq H$ or $M \not\subseteq H$ No solution if $M \not\subseteq H \wedge T \cup H \not\models M$ No solution if $\forall S : S \neq \emptyset \wedge S \subseteq H \rightarrow T \cup S$ inconsistent if $M \subseteq H$ then M is a solution if $M \not\subseteq H \wedge T \cup H \models M \wedge T \cup H$ consistent then H is a solution If M is solution, then correct, no guarantee otherwise	$M \subseteq H$ or $M \not\subseteq H$ No solution if $T \cup M$ inconsistent No solution if $M \not\subseteq H \wedge T \cup H \not\models M$ No solution if $\forall S : S \neq \emptyset \wedge S \subseteq H \rightarrow T \cup S$ inconsistent if $T \cup M$ consistent $\wedge$ $M \subseteq H$ then M is a solution if $T \cup H$ consistent $\wedge$ $M \not\subseteq H \wedge T \cup H \models M$ then H is a solution If M is solution, then correct (but not $T \cup M$), no guarantee otherwise
No Expert	$M \subseteq H$ M is solution If M is solution, then correct, no guarantee otherwise	$M \subseteq H$ M is solution iff $T \cup M$ consistent If M is solution, then correct (but not $T \cup M$), no guarantee otherwise

M / H	Missing + Wrong: T correct	Missing + Wrong: T not correct
Complete Knowledge	$M \not\subseteq H$ No solution	$M \not\subseteq H$ No solution if $T \cup M$ inconsistent No solution if $T \cup H \not\models M$ No solution if $\forall S : S \neq \emptyset \wedge S \subseteq H \rightarrow T \cup S$ inconsistent if $T \cup H$ consistent $\wedge$ $T \cup H \models M$ then H is a solution The solutions are not correct
Partial-Correct	$M \not\subseteq H$ No solution	$M \not\subseteq H$ No solution if $T \cup M$ inconsistent No solution if $T \cup H \not\models M$ No solution if $\forall S : S \neq \emptyset \wedge S \subseteq H \rightarrow T \cup S$ inconsistent if $T \cup H$ consistent $\wedge$ $T \cup H \models M$ then H is a solution The solutions are not correct
Wrong	$M \subseteq H$ or $M \not\subseteq H$ No solution if $T \cup M$ inconsistent No solution if $M \not\subseteq H \wedge T \cup H \not\models M$ No solution if $\forall S : S \neq \emptyset \wedge S \subseteq H \rightarrow T \cup S$ inconsistent if $T \cup M$ consistent $\wedge$ $M \subseteq H$ then M is a solution if $M \not\subseteq H \wedge T \cup H \models M \wedge T \cup H$ consistent then H is a solution The solutions are not correct	$M \subseteq H$ or $M \not\subseteq H$ No solution if $T \cup M$ inconsistent No solution if $M \not\subseteq H \wedge T \cup H \not\models M$ No solution if $\forall S : S \neq \emptyset \wedge S \subseteq H \rightarrow T \cup S$ inconsistent if $T \cup M$ consistent $\wedge$ $M \subseteq H$ then M is a solution if $T \cup H$ consistent $\wedge$ $M \not\subseteq H \wedge T \cup H \models M$ then H is a solution The solutions are not correct
No Expert	$M \subseteq H$ M is solution iff $T \cup M$ consistent The solutions are not correct	$M \subseteq H$ M is solution iff $T \cup M$ consistent The solutions are not correct

Table 1: Different combinations of cases for T, H and M.

Table 1 shows the properties for T, H, M and their combinations. For each combination we give information about the relationship between M and H, the existence of solutions and the correctness of the solutions. Here, we summarize the findings.

An ideal situation is the case where the domain expert has complete knowledge (H contains all correct is-a relations and no others) and T and M contain only correct is-a relations. In this case, $M \subseteq H$. Further, M is a solution and all solutions are correct.

For any case where $T \cup M$ is inconsistent, there is no solution. Indeed, for any solution S we have that $T \cup S \models M$ and thus $T \cup S$ would not be consistent.

In the cases where M contains wrong is-a relations, there may be no solutions. If there are solutions, these are not correct. Further, correctness of solutions is only guaranteed when M does not contain wrong is-a relations and H represents complete knowledge or partial-correct.

There are no solutions if $T \cup S$ is inconsistent for every non-empty subset S of H.

If $M \subseteq H$ and $T \cup M$ is consistent, then M is a solution. If $M \not\subseteq H$, $T \cup H$ is consistent and $T \cup H \models M$, then H is a solution.

In the case of no expert ($H = \{E_i \dot{\sqsubseteq} F_i \mid E_i, F_i \in C\}$) we have that $M \subseteq H$ and all is-a relations are allowed in the solution. Therefore, if $T \cup M$ is consistent, then M is a solution, otherwise there is no solution. However, as there is no domain expert, there is no guarantee that any solution other than M is correct. Further, in the cases where M contains wrong is-a relations, M is a solution, but not correct. As there is no validation, only logical consistency can be guaranteed, but no correctness.

3 Solutions with preference criteria

There can be many solutions for a GTAP and, as explained in Section 1, not all solutions are equally interesting. Therefore, we propose two preference criteria on the solutions.

Definition 2 (Subset Minimality) *A solution S to the GTAP (T, C, H, M) is said to be subset minimal iff there is no proper subset $S' \subsetneq S$ such that S' is a solution. The set of all subset minimal solutions is denoted as $\mathcal{S}_{min}(T, C, H, M)$.*

Examples of subset minimal solutions for $\mathcal{P}_1$ in Figure 2 are {limb-joint $\dot{\sqsubseteq}$ joint} and {hinderlimb-joint $\dot{\sqsubseteq}$ joint, forelimb-joint $\dot{\sqsubseteq}$ joint}.

Assuming there exist solutions, the answer to the existence problem for subset-minimal solutions is yes, if and only if $T \cup H \models M$. To find a solution S, we can start from H, and remove the is-a relations h stepwise, such that $T \cup H \setminus h \models M$ holds. The process continues until no is-a relation can be removed. Thus if the entailment problem for the underlying ontology is tractable, finding a solution can be done in polynomial time. This is indeed the case for the is-a taxonomy.

The second criterion prefers solutions that imply more information.

Definition 3 (More Informative) *Let S and S' be two solutions to the GTAP (T, C, H, M). S is said to be* more informative *than S' iff $T \cup S \models T \cup S'$ and there exists a ψ such that $T \cup S \models \psi$ and $T \cup S' \not\models \psi$. Further, we say that S is* equally informative *as S' iff $T \cup S \models S'$ and $T \cup S' \models S$.*

Consider two solutions to $\mathcal{P}_1$ in Figure 2, S = {limb-joint $\dot{\sqsubseteq}$ joint} and S'={hinderlimb-joint $\dot{\sqsubseteq}$ joint, hand-joint $\dot{\sqsubseteq}$ joint}. S is more informative than S' as T $\cup$ S entails limb-joint $\dot{\sqsubseteq}$ joint in addition to everything that T $\cup$ S' entails.

Definition 4 (Semantic Maximality) *A solution S to the GTAP (T, C, H, M) is said to be semantically maximal iff there is no solution S' which is more informative than S. The set of all semantically maximal solutions is denoted as $\mathcal{S}^{max}(T, C, H, M)$.*

Analogous to the subset minimality, assuming the existence of GTAP solutions, the answer to the existence problem for a semantically maximal solution is yes, if and only if $T \cup H \models M$ holds. Moreover, in the case where $M \subseteq H$, and $T \cup H$ is consistent, H is a semantically maximal solution.

In practice, both of the above two criteria are desirable. However, only with the semantic maximality we might obtain a solution with redundancy. Although subset minimality does not yield redundancy, there is no guarantee that the solution is the most informative. In the following we propose definitions on solutions by combining these criteria. There are diverse interpretations for the combination of subset minimality and semantic maximality, depending on what kind of priority we assign for the single preferences. A first interpretation implies a higher priority on subset minimality than the semantic maximality. As the second interpretation, higher priority for semantic maximality can be assigned to subset minimality. In the third interpretation, the skyline-style interpretation, we treat both preferences equally and the chosen solution is such that there does not exist another solution which is preferable on both criteria.

Definition 5 (Combining with priority for subset minimality) *A solution S to the GTAP (T, C, H, M) is said to be minmax optimal iff S is subset minimal and there does not exist another subset minimal solution S' such that S' is more informative than S. The set of all minmax optimal solutions is denoted as $\mathcal{S}^{max}_{\mathbf{min}}(T, C, H, M)$.*

Lemma 1. $\mathcal{S}^{max}_{\mathbf{min}}(T, C, H, M) \subseteq \mathcal{S}_{min}(T, C, H, M)$

As an example, $\{$limb-joint $\dot{\sqsubseteq}$ joint$\}$ is a minmax optimal solution for $\mathcal{P}_1$, while $\{$hinderlimb-joint $\dot{\sqsubseteq}$ joint, forelimb-joint $\dot{\sqsubseteq}$ joint$\}$ is a minmax optimal solution for $\mathcal{P}_2$.

The existence problem is equivalent to the existence problem of the subset minimal solutions, i.e., there exists a subset minimal solution if and only if there exists a minmax optimal solution. On the other hand, finding a minmax optimal solution tends to be a harder problem. One naive method is first collecting all the subset minimal solutions, then removing those which are less informative. Obviously this is intractable, because theoretically there could be an exponential number of subset minimal solutions already.

In practice, minmax optimal solutions ensure fewer is-a relations to be added, thus avoiding redundancy. This is desirable if the domain expert would prefer to look at as small solutions as possible. The disadvantage is that there may be redundant relations that are correct and not be derivable when they are not added.

Definition 6 (Combining with priority for semantic maximality) *A solution S to the GTAP (T, C, H, M) is said to be maxmin optimal iff S is semantically maximal and there does not exist another semantically maximal solution S' such that S' is a proper subset of S. The set of all maxmin optimal solutions is denoted as $\mathcal{S}^{\mathbf{max}}_{min}(T, C, H, M)$.*

Lemma 2. $\mathcal{S}^{\mathbf{max}}_{min}(T, C, H, M) \subseteq \mathcal{S}^{max}(T, C, H, M)$

As an example, $\{$limb-joint $\dot{\sqsubseteq}$ joint, autopod-joint $\dot{\sqsubseteq}$ limb-joint$\}$ is a maxmin optimal solution for $\mathcal{P}_1$.

Analogous to the case of minmax optimal, the existence problem of maxmin optimal is equivalent to the existence problem of the semantic maximal solutions. Moreover, if H is a semantically maximal solution, finding a maxmin optimal solution S can be done by starting from H, and stepwise removing the is-a relations h such that $T \cup S \setminus h \models H$ holds. Intuitively, the goal is to remove the redundant relations in H. Of course there might be multiple maxmin optimal solutions in this regard, but finding one such a solution is tractable as long as the reasoning task for the underlying logic is tractable.

The advantage of the maxmin optimal semantics is that a maximal body of correct information is added to the ontology. If the domain expert would prefer to look at as informative solutions as possible without (set) redundancy, maxmin optimal solutions is preferable than the minmax optimal solutions. This conclusion can even be strengthened from the efficiency point of view, as finding a maxmin optimal solution is more efficient than finding a minmax optimal one. The disadvantage is that more relations need to be validated.

For the skyline interpretation, we consider the subset minimality and the semantic maximality as two dimensions for a solution S. S is skyline optimal if it is not dominated by any other solution. A solution dominates another solution if it is as good or better in all dimensions and better in at least one dimension. Therefore regarding the above two dimensions we define that a solution S dominates another solution S' if one of the following conditions is fulfilled:

1. $S \subsetneq S'$ and S is more informative than S', or
2. $S = S'$ and S is more informative than S', or
3. $S \subsetneq S'$ and S is equally informative as S'.

It is easy to verify that condition 1 and 2 can never be fulfilled, due to the monotonicity property of the entailment. Therefore, a solution S dominates another solution S' if and only if condition 3 is fulfilled. Accordingly, we have the definition for the skyline optimality as follows.

Definition 7 (Skyline optimal) *A solution S to the GTAP (T, C, H, M) is said to be skyline optimal iff there does not exist another solution S' such that S' is a proper subset of S and S' is equally informative as S. The set of all skyline optimal solutions is denoted as $\mathcal{S}^{max}_{min}(T, C, H, M)$.*

Skyline optimal is a relaxed criterion. It requires subset minimality for some level of informativeness. It comprises all the subset minimal solutions – which in turn comprises all the minmax optimal solutions – and all the maxmin optimal solutions. This relationship can be easily verified.

Lemma 3. $\mathcal{S}_{min}(T, C, H, M) \cup \mathcal{S}^{\mathbf{max}}_{min}(T, C, H, M) \subseteq \mathcal{S}^{max}_{min}(T, C, H, M)$.

As an example, M in Figure 2 is a skyline optimal solution for $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$ and $\mathcal{P}_4$. All previous examples for subset mininal, minmax optimal and maxmin optimal solutions are also skyline optimal solutions. However, there are semantically maximal solutions that are not skyline optimal. For instance, $\{$hinderlimb-joint $\dot{\sqsubseteq}$ joint, forelimb-joint $\dot{\sqsubseteq}$ joint, hand-joint $\dot{\sqsubseteq}$ joint$\}$ is a semantically maximal solution for $\mathcal{P}_2$, but it is not skyline optimal as its subset $\{$hinderlimb-joint $\dot{\sqsubseteq}$ joint, forelimb-joint $\dot{\sqsubseteq}$ joint$\}$ is equally informative.

4 Debugging in practice

4.1 General observations

A system for repairing the missing is-a structure in ontologies, takes as input the ontology T and a set of is-a relations to repair M. C is implicit and can be computed using

T. Further, the system should be used by a domain expert who validates is-a relations (H)[3]. In general, however, when starting a debugging session, we do not know the properties of T, M and H. Further, H represents the knowledge about is-a relations from the domain expert, but is normally not available beforehand, but only through interaction of the domain expert with the debugging system. This means that even in the situations where H is a solution, this does not readily provide us a solution in practice. It also means that one cannot just take subsets of H and check whether they are solutions.

Table 1 provides us with some guidelines for the development and the use of debugging systems. First, it is clear that we prefer an all-knowing expert. The second best case for obtaining correct solutions is the partial-correct expert. As discussed in Section 2, this could be approximated by using multiple domain experts and a skeptical approach.

If there are wrong is-a relations in M, there will be no solution or solutions that are not correct. The repaired ontology will contain incorrect is-a relations. Therefore, the expert should validate M at the beginning of the debugging session. Those is-a relations which are identified to be incorrect should be removed from M.[4] Another advantage of the validation is that, after validation we have that $M_{validated} \subseteq H$.

Further, as we do not know whether T is correct according to the domain or not, it should be checked whether $T \cup M_{validated}$ is consistent. If not, then there are no solutions. Otherwise, we know that $M_{validated}$ is a solution. When we remove the redundancy from $M_{validated}$, then we also have a subset minimal solution. This solution could then be used as a basis for finding more informative solutions. The difficulty is in finding subsets S of H (which is not available) such that $T \cup S$ is consistent.

4.2 Lessons for an existing system

The system in [13] allows debugging the is-a structure of and mappings between taxonomies in a taxonomy network. The input to the system is an ontology network. In this discussion we focus on one of the ontologies in the network (T and thus also C). The debugging workflow consists of three phases: (1) detection (generation of M), (2) validation of M and (3) repair (solving the GTAP problem). The domain expert is involved in the validation of M as well as in phase 3 for validation of possible solutions (S). The domain expert can switch between the different phases at any time. The system was used in a real case for the Swedish National Food Agency [12] and in several experiments with ontologies from the Ontology Alignment Evaluation Initiative [19].

Although the system allows to switch between the different phases, in all our experiments we started with validating M, which is as suggested by our analysis in Section 4.1. If M contained wrong is-a relations, we used semantic debugging techniques to repair these. This allowed us to remove incorrect is-a relations in T. When all the wrong is-a relations are repaired and removed from M, we obtain a new $M_{validated}$. If the domain expert validated M in a correct way, we are in a situation in the upper part of Table 1. The is-a relations in $M_{validated}$ are then repaired. When they are repaired using

[3] If there would be no expert, as shown in Table 1, in the best case M could be a correct solution, but there is no guarantee for solutions. We do not discuss this case further in this section.

[4] Depending on the detection method to generate M, the wrong is-a relations in M may lead to other debugging opportunities for semantic defects (e.g., [13]).

solutions that are more informative than $M_{validated}$, then new knowledge is added to the network and a new round of detection was started, possibly leading to the detection, validation and repair of new is-a relations.

Initially, $M_{validated}$ is added to the ontology. This means that we start with a least informative solution. When removing redundancy from $M_{validated}$, it is also a subset minimal solution. Then, the system tries to generate more informative solutions. For this, the missing is-a relations are repaired one at the time. For each missing is-a relation m_i a set of is-a relations R_i is computed that guarantees that $T \cup \{r_i\} \models m_i$ for each $r_i \in R_i$. Thus, for each missing is-a relation, at most one is-a relation is added to the ontology. By removing redundancy subset minimal solutions can be guaranteed. Further, for each missing is-a relation on its own semantically maximal solutions are generated with the extra conditions that only one is-a relation is used for repairing and no unnecessary equivalences ($\ll_{SH}$ in [21]) are introduced in the ontology.

One immediate consequence of our analysis is that we should allow a domain expert to choose several elements of each R_i. This is an easy extension to the system that would provide more informative solutions. Another consequence is that it would be advantageous to allow a domain expert to deal with a previously repaired is-a relation again, when new knowlegde was added to the ontology. New more informative solutions may be found. Further, there should be a way for domain experts to add new is-a relations that do not occur within the repairing process.

An interesting observation during the debugging described in [12] was that the domain experts changed their mind about the correctness of some is-a relations after debugging some other is-a relations. This means that H may actually change during a session, and we may move upwards in Table 1.

5 Related Work

Repairing missing is-a relations. There is not much work on the repairing of missing is-a structure. In [21, 20] this was addressed in the setting of taxonomies where the problem as well as some preference criteria were defined. Further, an algorithm was given for finding a solution to the repairing problem and an implemented system was proposed. A later version of that system was then used for debugging ontologies related to a project for the Swedish National Food Agency [12]. The system was further extended to deal with missing and wrong is-a relations and mappings [19] and integrated with ontology alignment [13]. In [18] the problem was formalized as an abduction problem and an algorithm was given for finding solutions for $\mathcal{ALC}$ acyclic terminologies.

TBox abduction. Except for [18] in which GTAP without H was defined, there is no other work yet on GTAP. There is some work on TBox abduction. [11] proposes an automata-based approach to TBox abduction using abducibles. It is based on a reduction to the axiom pinpointing problem which is then solved with automata-based methods.

Related topics. There is work that addresses related topics but not directly the problem that is addressed in this paper. Regarding *detecting missing is-a relations* there is much work on finding relationships between terms in the ontology learning area [2]. Further, there is work on finding is-a relations based on different kinds of patterns (e.g., [9, 4]). When the ontology is part of a network of ontologies connected by mappings,

knowledge itrinsic to the ontology network can be used to detect missing is-a relations using logical derivation [21, 12]. These approaches, in general, do not detect *all* missing is-a relations. There is much work on *debugging semantic defects*. Most of the work on debugging semantic defects aims at identifying and removing logical contradictions from an ontology (e.g., [8, 26, 16, 10, 24, 27, 1, 23]). In [22, 28, 25, 14, 15] the setting is extended to repairing ontologies connected by mappings. Further, there is some work on *abductive reasoning in description logics*. In [7] four different abductive reasoning tasks are defined - concept, ABox, TBox and knowledge base abduction. Concept abduction deals with finding sub-concepts. Abox abduction deals with retrieving instances that, when added to the knowledge base, allow the entailment of a desired ABox assertion. Knowledge base abduction includes both ABox and TBox abduction. Most existing approaches focus on ABox [17, 6] and concept abduction [3, 5].

6 Conclusion

In this paper we formalized repairing missing is-a structure in ontologies as an abduction problem. We defined properties for the ontology, the set of is-a relations to repair and the domain expert, as well as preference criteria on solutions and discussed the influences of these properties and criteria on the existence of solutions for the abductive problem. We also discussed the consequences of our analyses for the development and use of debugging systems. One direction for future work is to analyze the complexity of the decision problems for different knowledge representation languages. Further, we want to investigate in algorithms that satisfy the preference criteria for different languages and that can be used in practice in a debugging system.

References

1. S Bail, B Parsia, and U Sattler. Declutter your justifications: Determining similarity between OWL explanations. In *1st International Workshop on Debugging Ontologies and Ontology Mappings*, pages 13–24, 2012.
2. Ph Cimiano, P Buitelaar, and B Magnini. *Ontology Learning from Text: Methods, Evaluation and Applications*. IOS Press, 2005.
3. S Colucci, T Di Noia, E Di Sciascio, F Donini, and M Mongiello. A uniform tableaux-based approach to concept abduction and contraction in ALN. In *International Workshop on Description Logics*, pages 158–167, 2004.
4. O Corcho, C Roussey, L M Vilches, and I Pérez. Pattern-based OWL ontology debugging guidelines. In *Workshop on Ontology Patterns*, pages 68–82, 2009.
5. F Donini, S Colucci, T Di Noia, and E Di Sciasco. A tableaux-based method for computing least common subsumers for expressive description logics. In *21st International Joint Conference on Artificial Intelligence*, pages 739–745, 2009.
6. J Du, G Qiand Y-D Shen, and J Pan. Towards practical Abox abduction in large OWL DL ontologies. In *25th AAAI Conference on Artificial Intelligence*, pages 1160–1165, 2011.
7. C Elsenbroich, O Kutz, and U Sattler. A case for abductive reasoning over ontologies. In *OWL: Experiences and Directions*, 2006.
8. P Haase and L Stojanovic. Consistent Evolution of OWL Ontologies. In *2nd European Semantic Web Conference*, pages 182–197. 2005.

9. M Hearst. Automatic acquisition of hyponyms from large text corpora. In *14th International Conference on Computational Linguistics*, pages 539–545, 1992.
10. M Horridge, B Parsia, and U Sattler. Laconic and precise justifications in OWL. In *7th International Semantic Web Conference*, pages 323–338, 2008.
11. T Hubauer, S Lamparter, and M Pirker. Automata-based abduction for tractable diagnosis. In *International Workshop on Description Logics*, pages 360–371, 2010.
12. V Ivanova, J Laurila Bergman, U Hammerling, and P Lambrix. Debugging taxonomies and their alignments: the ToxOntology - MeSH use case. In *1st International Workshop on Debugging Ontologies and Ontology Mappings*, pages 25–36, 2012.
13. V Ivanova and P Lambrix. A unified approach for aligning taxonomies and debugging taxonomies and their alignments. In *10th Extended Semantic Web Conference*, pages 1–15, 2013.
14. Q Ji, P Haase, G Qi, P Hitzler, and S Stadtmuller. RaDON - repair and diagnosis in ontology networks. In *6th European Semantic Web Conference*, pages 863–867, 2009.
15. E Jimenez-Ruiz, B Cuenca Grau, I Horrocks, and R Berlanga. Ontology Integration Using Mappings: Towards Getting the Right Logical Consequences. In *6th European Semantic Web Conference*, pages 173–187, 2009.
16. A Kalyanpur, B Parsia, E Sirin, and J Hendler. Debugging Unsatisfiable Classes in OWL Ontologies. *Journal of Web Semantics*, 3(4):268–293, 2006.
17. S Klarman, U Endriss, and S Schlobach. Abox abduction in the description logic ALC. *Journal of Automated Reasoning*, 46:43–80, 2011.
18. P Lambrix, Z Dragisic, and V Ivanova. Get my pizza right: Repairing missing is-a relations in ALC ontologies. In *2nd Joint International Semantic Technology Conference*, pages 17–32, 2012.
19. P Lambrix and V Ivanova. A unified approach for debugging is-a structure and mappings in networked taxonomies. *Journal of Biomedical Semantics*, 4:10, 2013.
20. P Lambrix and Q Liu. Debugging the missing is-a structure within taxonomies networked by partial reference alignments. *Data & Knowledge Engineering*, 2013.
21. P Lambrix, Q Liu, and H Tan. Repairing the Missing is-a Structure of Ontologies. In *4th Asian Semantic Web Conference*, pages 76–90, 2009.
22. C Meilicke, H Stuckenschmidt, and A Tamilin. Repairing Ontology Mappings. In *22th National Conference on Artificial Intelligence*, pages 1408–1413, 2007.
23. T Nguyen, R Power, P Piwek, and S Williams. Measuring the understandability of deduction rules for OWL. In *1st International Workshop on Debugging Ontologies and Ontology Mappings*, pages 1–12, 2012.
24. R Penaloza and B Sertkaya. On the complexity of axiom pinpointing in the EL family of description logics. In *12th International Conference on Principles of Knowledge Representation and Reasoning*, pages 280–289, 2010.
25. G Qi, Q Ji, and P Haase. A Conflict-Based Operator for Mapping Revision. In *8th International Semantic Web Conference*, pages 521–536, 2009.
26. S Schlobach. Debugging and Semantic Clarification by Pinpointing. In *2nd European Semantic Web Conference*, pages 226–240, 2005.
27. K Shchekotykhin, G Friedrich, Ph Fleiss, and P Rodler. Interactive ontology debugging: Two query strategies for efficient fault localization. *Journal of Web Semantics*, 12-13:88–103, 2012.
28. P Wang and B Xu. Debugging ontology mappings: a static approach. *Computing and Informatics*, 27:21–36, 2008.

Antipattern Detection: How to Debug an Ontology without a Reasoner

Catherine Roussey[1] and Ondřej Zamazal[2]

[1] Irstea, 24 Av. des Landais, BP 50085, 63172 Aubiére, France
catherine.roussey@irstea.fr
[2] Knowledge Engineering Group, University of Economics Prague, Czech Republic
ondrej.zamazal@vse.cz

Abstract. In ontology design, an Ontology Design Pattern (ODP) is a modeling solution to a recurrent ontology design problems. As opposed to ODP, antipatterns are bad modeling practices. This paper deals with detection of antipattern for debugging purpose of huge ontologies. We focus on the detection of the Onlyness Is Loneliness (OIL) antipattern. We propose an antipattern detection method based on ontology transformations and SPARQL queries. This approach does not need reasoner to detect antipattern. Our method detects candidates of OIL antipattern. These candidates localize class definitions where OIL occurrences can appear. This enables to draw ontology developer's attention to avoid errors during ontology development process. We conduct some experiments to detect OIL antipattern in an OWL ontology corpus obtained from the Watson ontology search engine.

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

1 Introduction

In ontology design an Ontology Design Pattern (ODP) is a modeling solution to a recurrent ontology design problems as defined in [7]. As opposed to ODP, antipatterns are bad modeling practices implemented in ontologies. Antipatterns produce the side effect of inferring wrong or undesired knowledge or of preventing the capabilities to infer the desired knowledge [16]. First, ontology antipatterns might help with guiding and training new ontology developers (educational purpose). Second, antipatterns can be directly used for ontology design purpose since ontology designers could take advantage of antipattern detection using some ontology editor during ontology development. Finally, detection of ontology antipatterns can contribute to ontology quality assessment.

In our previous work we first published our catalogue of antipatterns [6]. We have also provided some recommandations about ontologies repairing processes based on the antipattern detection. These patterns basically resulted in unsatisfiable classes or modeling errors due to the misuse or misunderstanding of Description Logics (DL) expressions. In [18] we proposed a general approach of

antipattern detection based on SPARQL queries which was applied on selected antipatterns from the catalogue [6].

The motivation for the work presented in this paper was twofold. On the one side, in [18] we show that each antipattern needs its own specific detection method. On the other side, reasoners might have troubles to tackle ontologies with complex axioms. But, reasoner output is usually prerequisite for a detection of complex antipatterns. Therefore, here we come up with an extension of our general detection method for one specific antipattern and in the situation that we cannot apply a reasoner. Instead of reasoner we apply ontology transformation pre-processing step and we evaluate it on one of the most complex antipatterns from our catalogue, *Onlyness Is Loneliness* (OIL) antipattern. Transformation have several goals: harmonization of ontology developer's implementation style, simulation of reasoner inferences and simplification of class definition axioms. It enables us to detect candidates of OIL antipattern. They are localized in class definitions where OIL occurrences can appear. Thus, they draw ontology developer's attention so that (s)he can avoid errors arised during ontology development process. We conduct some experiments to detect OIL antipattern in an OWL ontology corpus obtained from the Watson ontology search engine.[3]

The rest of the paper is structured as follows. Next section gives a brief overview of ontology design patterns and antipatterns for ontology development. Section 3 describes the OIL antipattern that is used to run our experiments. Next, Section 4 describes the detection method and our transformation rules. Section 5 describes the experiment setup and the results of the experimentation. Finally, Section 6 wraps up the paper by providing conclusions and future work.

2 Related Work

From Regarding educational purpose of bad modeling practices in DL, authors in [17] describe common difficulties in understanding of the logical meaning of expressions for newcomers to DL. Explicit antipatterns are then proposed in [6] presenting a catalogue of antipatterns based on DL expressions and proposing some recommendations on how to repair them.

To the best of our knowledge research in antipatterns are mainly connected to ontology debugging task. One of the earliest work was the OntoClean method [8], 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 source of antipatterns is [19], where authors proposed four terminological patterns applied on class names to detect possible errors along taxonomy. Next, the OntOlogy Pitfalls Scanner (OOPS) [15] enables an ontology developer to detect common pitfalls during the development of ontology.

There are several tools which can be used for antipattern detection. They are mostly available inside ontology editors and require the use of a reasoner to provide their justifications. For instance Pellint [5] focuses on the detection and

[3] `http://watson.kmi.open.ac.uk/`

repair of antipatterns to improve ontology reasoning performance. The Protégé Explanation Workbench [9] and SWOOP [12] provide justifications of inconsistencies in ontologies based on the output from DL reasoners. SWOOP has also a repair plug-in to help user during the debugging process [11]. However, using a reasoner for this purpose is not always possible, since in some big ontologies reasoners fail to provide any result [13]. Furthermore, the antipatterns repertory that these tools can detect is fixed.

Next, Ontology Pre-processor Language (OPPL) [10] enables pattern-based manipulation with ontologies. Authors of [14] describe an experiment using OPPL to detect ontology design patterns in a repository of biomedical ontologies. They also use transformations to harmonize the ontology, but their set of transformations only normalize 'syntactic sugar' conventions of OWL 2.

3 "Onlyness Is Loneliness" (OIL) Antipattern

One of the most common error made by ontology developer is captured by *Onlyness Is Loneliness (OIL) antipattern*. This antipattern can be manifested by one of the following sets of DL axioms:

$$C_3 \sqsubseteq \forall r.C_1; C_3 \sqsubseteq \forall r.C_2;\ \ Disj(C_1, C_2); \tag{1}$$
$$C_3 \equiv \forall r.C_1; C_3 \sqsubseteq \forall r.C_2;\ \ Disj(C_1, C_2); \tag{2}$$
$$C_3 \equiv \forall r.C_1; C_3 \equiv \forall r.C_2;\ \ Disj(C_1, C_2); \tag{3}$$
$$C_3 \sqsubseteq \forall r.C_1 \sqcap \forall r.C_2;\ \ Disj(C_1, C_2); \tag{4}$$
$$C_3 \equiv \forall r.C_1 \sqcap \forall r.C_2;\ \ Disj(C_1, C_2); \tag{5}$$

An OIL antipattern occurrence is defined by two disjoint classes C_1 and C_2 and a third class C_3 whose definition contains two universal restrictions using a property r. The first universal restriction expresses that instances of C_3 can only be linked with r to instances of C_1.[4] The second one expresses that instances of C_3 can only be linked with r to instances of C_2.

Based on our experience the origin of this error is that an ontology developer forgets that there is already a constraint about the class C_3 using an universal restriction and (s)he adds a new constraint about this class using another universal restriction. Moreover, one of these constraints can be inherited from any of the parent classes.

This error is already mentioned in several works such as [17] or [15]. In [15], the OIL antipattern is linked to the pitfall number 14, related to the misuse of universal restriction. Notice that even if the OIL antipattern is declared as Pitfall 14 in the catalogue of common pitfalls, the OntOlogy Pitfalls Scanner is not yet able to detect it.

[4] 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.

Detection of OIL antipattern is a difficult task (similarly to other antipatterns) due to various problems. First, antipatterns have various manifestations. For example, we present above 5 logical formulae related to OIL but we could imagine more formulae. Furthermore, 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 unnamed classes (or anonymous classes), e.g., $C \sqsubseteq (\exists R.X) \sqcap (\forall R.Y)$. In that case parts of antipattern can also be located at different places. Others can prefer to write short definitions. A class is defined by a set of atomic axioms,[5] e.g., $C \sqsubseteq \exists R.X; C \sqsubseteq \forall R.Y$. Each implementation style corresponds to a different logical formula of the OIL antipattern. Finally, we also have to consider that parts of the OIL antipattern can be asserted by the ontology developer or inferred by a reasoner.

4 OIL Antipattern Detection Method

General detection approach was presented in [18] based on SPARQL queries and reasoner output. However, none of those presented methods gives significant results for the OIL antipattern. Here, we describe a method aiming at detection of OIL antipattern in OWL ontologies. The main objective of this new method is to enable detection of OIL antipattern without reasoner output. As we experienced during debugging/repairing of complex ontologies (e.g. HydrOntology) by using the reasoner (e.g. Pellet)[6] and the Explanation Workbench tool,[7] applying reasoner for the antipattern detection is not always possible. It turns out that reasoner may fail to provide any results on complex ontologies. Generally, the more the number of repaired axioms increases, the more time reasoner needs to take in order to provide justifications for unsatisfiable classes.

This new method consists of two steps. First transformation rules are applied, see Section 4.1, and then SPARQL query (or, in general, SPARQL queries), see Section 4.2, is (are) executed using *PatOMat ontology pattern detection tool*, see Figure 1.[8] This tool is part of the PatOMat suite of tools, which is focused on the pattern detection in ontologies and their transformations. This detection tool is based on Jena 2.6.2 [1] and Pellet 2.0.1 [2], and enables the processing of a set of SPARQL queries over a set of ontologies in a batch. It produces 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.

Our transformations have several objectives:

[5] We defined an atomic axiom as a constraint (necessary condition $\sqsubseteq$ or sufficient condition $\equiv$) associated to a named class C using at most one DL 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.

[6] `http://owl.cs.manchester.ac.uk/explanation/`

[7] `http://owl.cs.manchester.ac.uk/explanation/`

[8] `http://owl.vse.cz:8080/DetectionTool/`

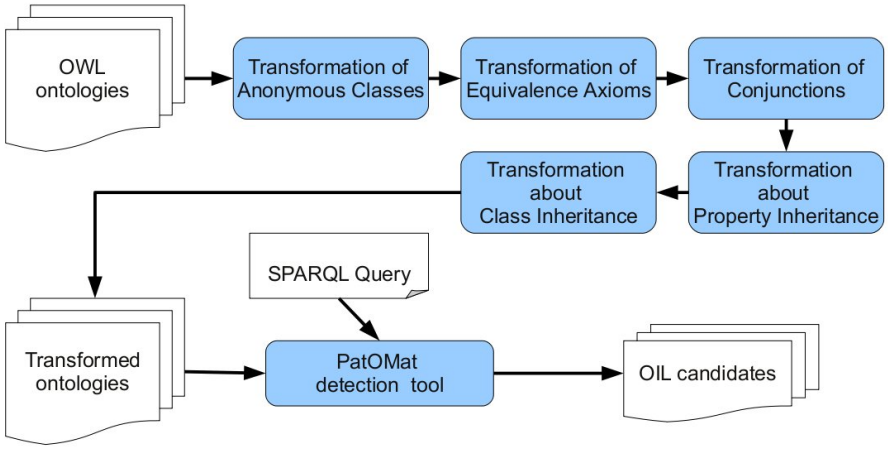

Fig. 1. Overview of the OIL detection method

- to decompose the class definition to simpler set of atomic axioms,
- to harmonize the different implementation styles of ontology developers,
- to simulate inferences in order to avoid applying a reasoner.

Our transformation rules only add new axioms and do not remove any original axioms from the ontology. The transformation rules have to be applied in the specific order. The rules are incremental, which means that the new axioms created by one transformation rule are already considered by a subsequent transformation rule.

According to [18], the use of a reasoner is mandatory to detect a disjoint axiom. In this previous work, we tried to detect OIL antipattern with asserted disjointness axioms. Since only few OIL occurrences were detected, we came up with the following hypothesis:

Hypothesis On the one side the disjoint axiom of the OIL antipattern is difficult to detect without a reasoner output and on the other side it turns out that asserted disjoint axioms do not help in detection. Thus, we limit the detection of OIL antipattern to the two first atomic axioms:

$$C_3 \sqsubseteq \forall r.C_1; C_3 \sqsubseteq \forall r.C_2; \tag{6}$$

We defined a class definition that contains this set of axioms as an OIL candidate.

4.1 Applied Transformation Rules

Here, we will explain our transformation rules one by one. For clarity purpose the transformation rules presented in this section are merely dedicated to the

OIL detection. For the general antipattern detection, there is a larger set of transformation rules.

TR AC: Transformation of Anonymous Classes TR AC consists of the following transformation rules:

$$C \sqsubseteq \forall r.(X \sqcup Y); \Rightarrow C \sqsubseteq \forall r.Or\_X\_Y; Or\_X\_Y \equiv X \sqcup Y; \quad (7)$$

$$C \sqsubseteq \forall r.(X \sqcap Y); \Rightarrow C \sqsubseteq \forall r.And\_X\_Y; And\_X\_Y \equiv X \sqcap Y; \quad (8)$$

$$C \sqsubseteq \forall r.(\exists s.Z); \Rightarrow C \sqsubseteq \forall r.Some\_S\_Z; Some\_S\_Z \equiv \exists s.Z; \quad (9)$$

$$C \sqsubseteq \forall r.(\forall s.Z); \Rightarrow C \sqsubseteq \forall r.Only\_S\_Z; Only\_S\_Z \equiv \forall s.Z; \quad (10)$$

$$C \equiv \forall r.(X \sqcup Y); \Rightarrow C \equiv \forall r.Or\_X\_Y; Or\_X\_Y \equiv X \sqcup Y; \quad (11)$$

...

The goal of this transformation is to name anonymous class in order to detect any antipattern in new named class definitions. Moreover, this transformation removes brackets and shortens class definitions. Some ontology developers write long definition of class using anonymous classes (or unnamed classes), e.g., as on the left-hand-side of formulae above. To simplify the query of OWL ontologies, class definitions must be transformed in this way. Ideally, class definitions are only composed of atomic axioms, so we need to remove anonymous classes, e.g. $(X \sqcup Y)$, $(X \sqcap Y)$ or $(\exists s.Z)$. The antipattern can be located in the anonymous class definition so we need to create a named class for each of them. In order to easily detect that these named classes come from the transformation rule we apply the following naming convention, e.g., And_X_Y in the case of $(X \sqcap Y)$.

In all, this transformation will be applied on each anonymous class in brackets so that new named class will be created and will replace original anonymous class in all axioms of the ontology. This new class will be defined by the anonymous class content and ideally correctly placed to the taxonomy, e.g. $And\_X\_Y \equiv X \sqcap Y$ should be a child class of X and of Y and $Or\_X\_Y \equiv X \sqcup Y$ should be a parent class of X and of Y.

TR EA: Transformation of Equivalence Axioms TR EA consists of the following transformation rule:

$$C_A \equiv C_B; \Rightarrow C_A \sqsubseteq C_B; C_B \sqsubseteq C_A; \quad (12)$$

The $\equiv$ symbol should be transformed into both-sided $\sqsubseteq$, i.e. we create two new axioms with the subClassOf relationship. This transformation is necessary because each antipattern can be written with a $\equiv$ or $\sqsubseteq$ symbol. Due to this transformation rule we can just consider atomic axioms of the form $C \sqsubseteq ...$ for antipattern detection.

In all, this transformation will be applied on all equivalence axioms according to which new two subsumptions (subClassOf) axioms will be added.

TR Conj: Transformation of Conjunctions TR Conj consists of the following transformation rules:

$$\left.\begin{array}{l} C \sqsubseteq \forall r.X_1 \sqcap ... \\ \sqcap \exists r.X_2 \sqcap ... \\ \sqcap \leq x\ r.\top \sqcap ... \\ \sqcap \geq x\ r.\top \sqcap ... \\ \sqcap = x\ r.\top \sqcap ... \\ \sqcap X_i ... \\ \sqcap X_n \end{array}\right\} \Rightarrow \left\{\begin{array}{l} C \sqsubseteq \forall r.X_1; \\ C \sqsubseteq \exists r.X_2 \\ C \sqsubseteq \leq x\ r.\top; \\ C \sqsubseteq \geq x\ r.\top; \\ C \sqsubseteq = x\ r.\top; \\ ...; \\ C \sqsubseteq X_i; \\ ...; \\ C \sqsubseteq X_n; \end{array}\right. \tag{13}$$

An ontology developer has her/his own implementation style. Some of them prefer to write long axioms using conjunction of anonymous classes. On the contrary, others prefer to define lots of named classes in order to identify small part of knowledge. Depending on the implementation style of the ontology developer, the antipattern detection may be facilitated. For antipattern detection, we recommend to split all the long axioms into several atomic axioms.

In all, this transformation will be applied on conjunction of anonymous classes so that it will be split into its components, e.g. $C \sqsubseteq \forall r.X \sqcap \exists r.Y$ will generate two new axioms $C \sqsubseteq \forall r.X$ and $C \sqsubseteq \exists r.Y$.

TR PI: Transformation about Property Inheritance TR PI consists of the following transformation rule:

$$\begin{array}{r} r_1 \sqsubseteq r; \\ C_A \sqsubseteq \forall r_1.C_B; \Rightarrow C_A \sqsubseteq \forall r.C_B; \end{array} \tag{14}$$

The ontology developer can forget that (s)he has defined a property r_1 as a subproperty of another one, r. Thus, for each axiom using the subproperty r_1 we need to add a redundant axiom using the parent property r. This transformation is applied along the whole taxonomy of properties. The new axiom changes the semantics of the ontology. Thus, it should be added only if the class C_A is already defined by a universal restriction using the r property in order to check that there is no conflict between different universal restrictions using r.

TR CI: Transformation about Class Inheritance TR CI consists of the following transformation rule:

$$C_B \sqsubseteq C_A; C_A \sqsubseteq \forall r.Y; \Rightarrow C_B \sqsubseteq \forall r.Y; \tag{15}$$

The main purpose of this transformation is to simulate reasoning so that all subclasses inherit all (asserted) constraints from their parents. It is applied at the end of the transformation process because we want to inherit all previously added (due to the application of other transformation rules) axioms as well.

4.2 SPARQL Query

According to Figure 1 SPARQL query, by using PatOMat detection tool, is performed after application of transformations. The following query retrieves OIL candidates defined by equation 6:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?c3 ?r ?c1 ?c2
WHERE
{ ?c3 rdfs:subClassOf _:restrictionA1.
_:restrictionA1 rdf:type owl:Restriction.
_:restrictionA1 owl:onProperty ?r.
_:restrictionA1 owl:allValuesFrom ?c1.
?c3 rdfs:subClassOf _:restrictionB1.
_:restrictionB1 rdf:type owl:Restriction.
_:restrictionB1 owl:onProperty ?r.
_:restrictionB1 owl:allValuesFrom ?c2.
FILTER
( isURI(?c1) && isURI(?c2) && isURI(?c3) && ?c1!= ?c2 )
}
ORDER BY ?c3  ?r ?c1
```

The query finds a class $?c3$ defined by two universal restrictions using the same property $?r$. The first restriction is linked to the $?c1$ class, the second one is linked to the $?c2$ class. As shown in the filter part of the query, all the class variables ($?c3$, $?c1$, $?c2$) should be named classes.

OIL candidates localize class definitions where OIL antipattern occurrences can appear. Thus, these candidates draw ontology developer's attention to avoid errors arised during long ontology development process.

5 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 our detection method from Section 4 over this ontology corpus.

5.1 Building a Corpus of (Debuggable) OWL Ontologies

We have used the Watson API [3] 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 66 inconsistent ontologies. From these ontologies we have selected ontologies that cannot be classified by Pellet in a reasonable time. Using PatOMat, we queried and processed several sets of ontologies at the same time. We materialized the inferred axioms using Pellet and then we queried the resulting ontologies with a OIL query.[9] If Pellet could not classify the ontology or if the classifying process took more than 10 seconds, the ontology was used for our experimentation. [10]

Table 1 presents the ontologies used for our experimentation. For each ontology, this table indicates: its number of classes, information whether Pellet can classify it or not, the time which classification process by Pellet takes, its number of unsatisfiable classes. The last column indicates if the ontology was debugged or not. We use the Explanation Workbench tool [9] to debug these ontologies. This tool provides for each unsatisfiable class the minimal set of axioms in which the class is unsatisfiable. Regarding repairing process we do not simply remove problematic axioms. Our repairing process is rather collaborative task between a DL expert and a domain expert, i.e. the DL expert identifies what the domain expert wants to express and proposes the correct DL axioms capturing intended domain knowledge. When an ontology is debugged it means that we know the total number of OIL occurrences.

name	nr. of classes	Pellet	time execution	nr. of unsatisf. classes	debugged
HydrOntology	159	no		114	yes
Tambis	395	yes	27 s	112	yes
inconsistent 613	6	no		1	yes
inconsistent 623	11	no		1	yes
Open Cyc	2846	yes	43 s	1663	no
proteonic	401	yes	17 s	5	no
CSNCS	1274	yes	139 s	154	no

Table 1. the list of ontologies

5.2 Experiments

Evaluation of Antipattern Detection Precision The precision of the OIL candidate detection process was evaluated. One evaluator analyzed the SPARQL results and assigned to each OIL candidate one of the following values:

– *TI* (True positive Inconsistency): the evaluator is sure that the OIL candidate participates in the unsatisfiability of classes.

[9] This query is looking for the RDF representation of the OIL antipattern expressed by the equation number 1 in section 3

[10] All of these ontologies are available from [4].

– *UI* (Unknown Inconsistency): the evaluator is not able to take a decision.
– *FI* (False positive Inconsistency): the evaluator is sure that the OIL candidate does not participate in the unsatisfiability of classes.

5.3 Results of OIL candidate detection

Due to the symmetric property of OIL candidate[11], the query identifies an OIL candidate twice. Table 2 presents the query results and the evaluation results. The first column indicates the total number of OIL candidates found by the query. The next columns indicate the number of TI, UI and FI from the OIL candidates decided by the evaluator.

All the FI we found in the query results come from the fact that: (1) $c1$ and $c2$ are linked by a subClassOf relationship; (2) the $c1$ class is built by the TR AC transformation rule, and $c2$ class is one of the named classes used in the $c1$ definition. For example, there is the following OIL candidate in the results from the Hydrontology: $c3 = Poza; r = parte\_de; c1 = OR\_Lago\_Rio; c2 = Rio$.

set	nr. of results	nr. of TI	nr. of UI	nr. of FI
HydrOntology	84	44 (52 %)	0	41 (48 %)
Tambis	314	0	121 (39 %)	193 (61 %)
inconsistent 613	0	0	0	0
inconsistent 623	6	6 (100 %)	0	0
Open Cyc	0	0	0	0
Proteonic	610	0	65 (11 %)	545 (89 %)
CSNCS	21 914	0	0	21 914 (100 %)

Table 2. results of OIL candidate detection process

In the case of HydrOntology all the OIL antipattern occurrences were retrieved by our query. In the case of Tambis ontology, it does not contain any OIL antipattern occurrences. But thanks to the OIL candidates retrieved by the query, some dangerous classes definitions are detected because there exist two universal restrictions with sibling classes (these sibling classes are not inferred as disjoint classes so that there is no OIL antipattern). For example, we found: $c3 = gene-product; r = polymer-of; c1 = ribo-nucleotide; c2 = amino-acid$. Such results were tagged as UI by the evaluator.

The inconsistent 623 and 613 ontologies were debugged and they do not contain any OIL antipattern occurrences. These ontologies contain only one unsatisfiable class that is very difficult to debug due to the presence of transitive and inverse properties. In the case of the inconsistent 623 ontology, the detected OIL candidates identify one of the class involved in the class unsatisfiability.

[11] C_1 and C_2 can be switched in the equation 6.

Proteonic ontology was not yet debugged. The OIL candidates retrieved by the query identify some dangerous classes because there exist some OIL candidates, e.g. $c3 = collection; r = direct-part-of; c1 = collection; c2 = element$. These results were tagged as UI by the evaluator.

CSNCS ontology imports several ontologies. It imports the DUL ontology which describe classes using lots of universal restrictions. All the OIL candidates contains some DUL classes like Entity, Object, Social Object. This ontology also imports several large ontologies, e.g. the DOLCE Ultra Light ontology, which leads to many OIL candidates detected by the query.

Results from Table 2 are encouraging, because OIL candidates may be useful to detect error or dangerous class definitions. Using a list of transformations and a simple SPARQL query is sufficient for detecting complex antipattern like OIL one, even on large ontology that cannot be processed by a reasoner. Moreover during long development process it is useful to detect OIL candidates in order to localize dangerous class definitions where an error can often occur.

In this experimentation we have validated the fact that transformation processes and simple SPARQL queries are sufficient for detection some OIL antipattern without a reasoner. Note that in our previous work [18], the previous detection method based on reasoner output and using SPARQL queries were not able to detect any OIL occurrences at all. If we want to apply our method to other antipattern, we should define the adequate transformation rules and SPARQL queries.

Our immediate work will aim at presenting the SPARQL results so that the candidates where $c1$ and $c2$ classes are linked by a subClassOf relationship will be eliminated.

6 Conclusions and Future Work

In this paper we have shown how complex antipatterns such as OIL can be detected. Our detection method can work without a reasoner. It is based on ontology transformations and one SPARQL query. These transformations have several goals: harmonizing ontology developer implementation style, simulating reasoner inference and simplifying class definition axioms. Our method detects OIL antipattern candidates. These candidates localize class definition where OIL occurrences can appear. Thus, they draw ontology developer's attention to avoid errors during long ontology development process. We conduct some experiments to detect OIL antipattern in an OWL ontology corpus obtained from the Watson ontology search engine. Our future work will focus on the definition of new transformations to detect other complex antipatterns from our antipattern catalogue.

References

1. Apache jena. `http://jena.apache.org/` (2012)
2. Pellet: Owl 2 reasoner for java. `http://clarkparsia.com/pellet/` (2012)

3. Watson: Exploring the semantic web. http://watson.kmi.open.ac.uk/WS_and_API.html (2012)
4. web site related to our ontology antipattern detection methods. https://sites.google.com/site/ontologyantipattern (2012)
5. Clark, K.: Pellint: An ontology repair tool. http://weblog.clarkparsia.com/2008/07/02/pellint-an-ontology-repair-tool/ (2008)
6. Corcho, O., Roussey, C., Vilches Blázquez, L.M., Pérez, I.: Pattern-based OWL ontology debugging guidelines. In: Proceedings of WOP. pp. 68–82. CEUR-WS.org (2009)
7. Gangemi, A., Presutti, V.: Ontology design patterns. Handbook on Ontologies pp. 221–243 (2009)
8. Guarino, N., Welty, C.A.: Evaluating ontological decisions with OntoClean. Commun. ACM 45(2), 61–65 (2002)
9. Horridge, M., Parsia, B., Sattler, U.: Laconic and precise justifications in OWL. In: Proceedings of ISWC. pp. 323–338 (2008)
10. Iannone, L., Rector, A.L., Stevens, R.: Embedding knowledge patterns into OWL. In: Proceedings of ESWC. pp. 218–232 (2009)
11. Kalyanpur, A., Parsia, B., Sirin, E., Cuenca Grau, B.: Repairing unsatisfiable concepts in OWL ontologies. In: Proceedings of ESWC. pp. 170–184 (2006)
12. Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.: Debugging unsatisfiable classes in OWL ontologies. Journal of Web Semantics 3(4), 268–293 (2005)
13. Lehmann, J., Bhmann, L.: ORE - a tool for repairing and enriching knowledge bases. In: proceedings of ISWC. LNCS, vol. 6497- part 2, pp. 177–193. Springer, Shanghai, China (2010)
14. Mortensen, J., Horridge, M., Musen, M., Noy, N.: Modest use of ontology design patterns in a repository of biomedical ontologies. In: Proceedings of WOP. vol. 929. CEUR-WS.org, Boston, USA (2012)
15. Poveda-Villalon, M., Suarez-Figueroa, M.C., Gomez-Perez, A.: Validating ontologies with OOPS! In: proceedings of EKAW. LNCS, vol. 7603, pp. 267–281. Springer Berlin Heidelberg, Ireland (2012)
16. Presutti, V., Blomqvist, E., Daga, E., Gangemi, A.: Pattern-based ontology design. Ontology Engineering in a Networked World pp. 35–64 (2012)
17. Rector, A.L., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: OWL pizzas: Practical experience of teaching OWL-DL: common errors & common patterns. In: Proceedings of EKAW. pp. 63–81 (2004)
18. Roussey, C., Corcho, O., Svab-Zamazal, O., Scharffe, F., Bernard, S.: SPARQL-DL queries for antipattern detection. In: Proceedings of WOP. vol. 929. CEUR-WS.org, Boston, USA (2012)
19. Šváb-Zamazal, O., Svátek, V.: Analysing ontological structures through name pattern tracking. In: Proceedings of EKAW. pp. 213–228 (2008)

Ontology Adaptation upon Updates

Alessandro Solimando, Giovanna Guerrini

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
Università di Genova, Italy
name.surname@unige.it

Abstract. Ontologies, like any other model, change over time due to modifications in the modeled domain, deeper understanding of the domain by the modeler, error corrections, simple refactoring or shift of modeling granularity level. Local changes usually impact the remainder of the ontology as well as any other data and metadata defined over it. The massive size of ontologies and their possible fast update rate requires automatic adaptation methods for relieving ontology engineers from a manual intervention, in order to allow them to focus mainly on high-level inspection. This paper, in spirit of the *Principle of minimal change*, proposes a fully automatic ontology adaptation approach that reacts to ontology updates and computes sound reformulations of ontological axioms triggered by the presence of certain preconditions. The rule-based adaptation algorithm covers up to $\mathcal{SROIQ}$ DL.

1 Introduction and Motivations

Ontologies, like any other model, change over time and a revalidation of all data and metadata defined on top of the modified ontology is needed upon updates. Massive ontology size and fast update rate[1] call for automated support and adaptation algorithms. Despite the great attention devoted in the last ten years to ontology evolution [1, 7], to the best of our knowledge there are no proposals in the literature coping with ontology adaptation upon updates. With similar motivations, an adaptation algorithm for a subset of *SPARQL* queries (with expressivity equivalent to union of Conjunctive Queries) in response to ontology updates is proposed in [6]. *Protégé*[2], one of the most complete ontology frameworks, does not support any kind of adaptation w.r.t. ontology updates: when a concept or a role is deleted, all the axioms referring it are removed as well. Even if there are cases in which this behavior is acceptable (*e.g.*, error corrections), there are others for which it is detrimental, for instance a modification of the modeling granularity of the ontology. In this scenario, a sound reformulation of axioms by means of super/sub concepts or roles is not only desirable but usually manually performed by the modeler. Additionally, in Artificial Intelligence (*Belief Revision*), knowledge deletion usually follows the *Principle of Minimal*

[1] An example is the *Gene Ontology* (`http://www.geneontology.org/`), with $\sim 416K$ axioms and $\sim 40K$ entities, daily updated (statistics for data-version 2013-02-22).
[2] Available here: `http://protege.stanford.edu/`

Change [5], which suggests that the amount of lost information should be as minimal as possible. Given that ontologies do not necessarily (explicitly) include all their logical consequences, also the implicit knowledge should be taken into account, as well as explicit one (that is, ontology axioms).

While a set of basic ontology changes can be easily defined, it is impossible to identify a set of complex changes without fixing the granularity level, *i.e.*, updates expressed as arbitrarily complex graph patterns (see [10], Section 3.2.1). In this proposal we consider the basic updates proposed by [3]: addition, deletion and update of entities (concepts and roles). Given that adding or updating entities do not reduce knowledge, and that ontology consistency can be tested using ontology reasoners, our adaptation algorithm focuses only on entity deletions.

In this paper, we propose an algorithm that, given an ontology and an entity (concept or role) to delete, scans for an equivalent, a super and a sub-entity and tries to reformulate the axioms involving the entity in question, with a rule-based approach. Our reformulated axioms are a fraction of the *implicit knowledge* of the ontology under update that would be lost by deleting all of the axioms involving the removed entity. An alternative would be to compute the closure (that is, complete inference of implicit knowledge) for the ontology prior to entity deletion. Due to its high computational cost and possible non-finiteness of the result, a suboptimal but less expensive approach is preferable for our target scenario, that is interactive modeling.

Even if the adaptation algorithm is completely automatic, it may not always be aligned with the modeler's intention. For this reason, the present proposal has to be intended as an optional feature. When activated, it provides a preview of the changes to show the automatic adaptation effects. On this basis, the modeler can accept or ignore the proposed changes. In addition, a straightforward extension could be the possibility, for the modeler, to select the equivalent (resp. sub/super) entity for the reformulation, when different alternatives are available.

The contribution of the present paper can be summarized as follows: an automatic adaptation algorithm supporting up to $\mathcal{SROIQ}$ expressivity, its correctness proof, and temporal complexity analysis (Section 3), an experimental evaluation of the percentage of adaptable entities and axioms on a dataset of real ontologies (Section 4). First, DL basics are introduced (Section 2), and the paper concludes discussing future work (Section 5).

2 Preliminaries

Our proposal covers up to $\mathcal{SROIQ}$ Description Logic (DL), on top of which the *Ontology Web Language* (OWL2) [11] is defined. The notations and definitions used in this section are borrowed from [4]. An ontology is defined by a set of axioms and a set of entity names (signature), composed by three disjoint subsets: $N_{\mathcal{R}}$ for role names, $N_{\mathcal{I}}$ for individual names, $N_{\mathcal{C}}$ for concept names. These entities are defined by means of expressions. We have *Role expressions* $\boldsymbol{R} ::= U \mid N_{\mathcal{R}} \mid N_{\mathcal{R}}^{-}$, and *Concept expressions* $\boldsymbol{C} ::= N_{\mathcal{C}} \mid (\boldsymbol{C} \sqcup \boldsymbol{C}) \mid (\boldsymbol{C} \sqcap \boldsymbol{C}) \mid \neg \boldsymbol{C} \mid \top \mid \bot \mid \exists \boldsymbol{R}.\boldsymbol{C} \mid \forall \boldsymbol{R}.\boldsymbol{C} \mid \geq_n \boldsymbol{R}.\boldsymbol{C} \mid \leq_n \boldsymbol{R}.\boldsymbol{C} \mid \exists \boldsymbol{R}.Self \mid \{N_{\mathcal{I}}\}$, with $n \geq 0$. For the

	Precondition	Rule
a.1	$C \equiv C'$, $C \in signature(axiom)$	$axiom \rightarrow axiom[C/C']$
a.2	$C \sqsubseteq D$	$E \equiv \exists R.C \rightarrow E \sqsubseteq \exists R.D$
a.3	$C \sqsubseteq D$	$E \equiv \geq_n R.C \rightarrow E \sqsubseteq \geq_n R.D$
a.4	$C \sqsubseteq D$	$E \equiv C \sqcup F \rightarrow E \sqsubseteq D \sqcup F$
a.5	$C \sqsubseteq D$	$E \equiv C \sqcap F \rightarrow E \sqsubseteq D \sqcap F$
a.6	$C \sqsubseteq D$	$E \equiv \neg C \rightarrow \neg D \sqsubseteq E$
a.7	$C \sqsubseteq D$	$C(a) \rightarrow D(a)$
a.8	$C \sqsubseteq D$	$E \equiv \forall R.C \rightarrow E \sqsubseteq \forall R.D$
a.9	$B \sqsubseteq C$	$E \equiv \leq_n R.C \rightarrow E \sqsubseteq \leq_n R.B$
a.10	$B \sqsubseteq C$	$E \equiv C \sqcup F \rightarrow B \sqcup F \sqsubseteq E$
a.11	$B \sqsubseteq C$	$E \equiv C \sqcap F \rightarrow B \sqcap F \sqsubseteq E$
a.12	$B \sqsubseteq C$	$E \equiv \neg C \rightarrow E \sqsubseteq \neg B$
a.13	$B \sqsubseteq C$	$C \sqsubseteq E \rightarrow B \sqsubseteq E$

Table 1. Adaptation rules for concept deletion $DEL(C)$, where $B, C, C', D, E, F \in N_{\mathcal{C}}$, $R \in N_{\mathcal{R}}$ and $a \in N_{\mathcal{I}}$.

semantics associated with nominals, role and concept expressions the reader may refer to [4]. The set of axioms of an ontology, denoted with $Axioms$, is defined as $Axiom ::= ABox \cup RBox \cup TBox$. The reader may refer to [4] also for a detailed description of the different available axioms for $\mathcal{SROIQ}$ DL, and to [9] for the definitions of ontology *interpretation* and ontology *satisfiability*. W.l.o.g. in the paper we will consider normalized ontologies in *Negation Normal Form* (NNF), with an application of *Structural Reduction* (SR), as shown in [9] (Subsection 5.3). SR introduces fresh concept names for (complex) concept expressions, thus letting us to easily refer to each concept expression by means of its associated concept name. Neither the SR nor the NNF are required for the application of our method. NNF, however, may increase the ratio of adapted axioms.

3 Algorithm

This section introduces the adaptation rules (Section 3.1), the rule-based adaptation algorithm (Section 3.2), the correctness proof for the given rules (Section 3.3), and the temporal complexity of the algorithm (Section 3.4).

3.1 Adaptation Rules

The adaptation rules are presented in Table 1 (rules for concepts) and Table 2 (rules for roles). We denote by $axiom[A/B]$ the alpha renaming of an axiom of entity A by entity B. A rule r is composed by a left hand side, $LHS(r)$, a right hand side, $RHS(r)$, and a precondition $prec(r)$. A rule is defined **applicable** iff $prec(r)$ is satisfied by at least one concept (resp. role). Given an ontology o and an entity e to delete, the LHS of a rule r is said to be **matching** iff an axiom in o exists that is equal to $LHS(r)$ modulo alpha renaming of C (resp. R)

	Precondition	Rule
b.1	$R \equiv R'$, $R \in signature(axiom)$	$axiom \rightarrow axiom[R/R']$
b.2	$Q \sqsubseteq R$	$T_0 \circ \ldots \circ T_m \circ R \circ T'_0 \circ \ldots \circ T'_p \sqsubseteq T$ $\rightarrow T_0 \circ \ldots \circ T_m \circ Q \circ T'_0 \circ \ldots \circ T'_p \sqsubseteq T$
b.3	$Q \sqsubseteq R$	$E \equiv \forall R.C \rightarrow E \sqsubseteq \forall Q.C$
b.4	$Q \sqsubseteq R$	$E \equiv \leq_n R.C \rightarrow E \sqsubseteq \leq_n Q.C$
b.5	$Q \sqsubseteq R$	$T \equiv R^- \rightarrow Q^- \sqsubseteq T$
b.6	$Q \sqsubseteq R$	$Disjoint(R,T) \rightarrow Disjoint(Q,T)$
b.7	$R \sqsubseteq S$	$R(a,b) \rightarrow S(a,b)$
b.8	$R \sqsubseteq S$	$E \equiv \exists R.C \rightarrow E \sqsubseteq \exists S.C$
b.9	$R \sqsubseteq S$	$E \equiv \exists R.Self \rightarrow E \sqsubseteq \exists S.Self$
b.10	$R \sqsubseteq S$	$E \equiv \geq_n R.C \rightarrow E \sqsubseteq \geq_n S.C$
b.11	$R \sqsubseteq S$	$T \equiv R^- \rightarrow T \sqsubseteq S^-$
b.12	$R \sqsubseteq S$	$T_0 \circ \ldots \circ T_q \sqsubseteq R \rightarrow T_0 \circ \ldots \circ T_q \sqsubseteq S$

Table 2. Adaptation rules for role deletion $DEL(R)$, where E, $C \in N_C$, Q, R, R', S, T, T_i, $T'_j \in N_R$, with $m, n, p, q \geq 0$, and $a, b \in N_I$.

with e, denoted with $LHS(r)[e]$. The **application** of an applicable rule r w.r.t. o and e rewrites any axiom of o matching $LHS(r)[e]$ into $RHS(r)[e']$, where e' is the selected entity for reformulation. It is worth noting that if a DL less expressive than $\mathcal{SROIQ}$ is adapted, only a subset of the rules will be applicable, depending on the axioms and constructors available. For instance, for basic $\mathcal{ALC}$ with *General Concept Inclusion* (*i.e.*, $C \sqsubseteq D$), rules a.3, a.9, b.2, b.4, b.5, b.9, b.10, b.11, b.12 are not applicable.

3.2 Adaptation Algorithm

Algorithm 1 presents the adaptation algorithm for ontology updates. It takes as input the entity e to be deleted and the ontology o it belongs to. By means of function *computePrec*, the set of axioms related to e is computed, as well as a triple p consisting of a (nondeterministically choosen) equivalent, a sub and a super entity, if any (line 3). For each axiom a having e in its signature (line 4), it tests if the axiom matches the left hand side of the rule (line 5). At this point, function $satisfies$ (line 6) checks if the current axiom is compatible with rule r and if the required element in p is not null. The reformulated axiom is inserted in o (line 7). Finally, all the axioms involving entity e are removed from o (line 8). Even if a preliminar classification phase is not required, it may increase the algorithm effectiveness. In what follows we give a toy example of ontology update, comparing the result of adaptation to classical deletion approach.

Example 1. Consider an ontology o consisting of these axioms and the obvious associated signature: $Human \equiv \exists eats.Food$, $Food(cheese)$, $Eater \equiv \forall eats.Food$, $\bot \equiv Plastic \sqcap Food$, $Uneatable \equiv \neg Eatable$, $Pizza \sqsubseteq Food$, $Food \sqsubseteq Eatable$. Deleting $Food$ concept from o with adaptation we obtain: $Human \sqsubseteq \exists eats.Eatable$, $Eatable(cheese)$, $Eater \sqsubseteq \forall eats.Eatable$, $Plastic \sqcap$

Algorithm 1 Ontology Update Adaptation

```
1: function OntoUpdateAdapt(Entity e, Ontology o)
2:     axioms = ∅
3:     p := ⟨eq, sub, sup⟩ ← computePrec(e, axioms, o)
4:     for a ∈ axioms do
5:         for r ∈ Rules . a = LHS(r)[e] do
6:             if satisfies(⟨a, e, e'⟩, prec(r)), e' ∈ {eq, sub, sup} then
7:                 Axioms(o) ← Axioms(o) ∪ {RHS(r)[e']}
8:             end if
9:         end for
10:    end for
11:    Axioms(o) ← Axioms(o) \ axioms
12: end function
13: function computePrec(Entity e, Set axioms, Ontology o)
14:    eq, sub, sup ← ϵ
15:    for a ∈ Axioms(o) . e ∈ signature(a) do
16:        axioms ← axioms ∪ {a}
17:        if eq, sub, sup ≠ ϵ then
18:            break
19:        end if
20:        if a = e ≡ e' or a = e' ≡ e then
21:            eq ← e'
22:        else if a = e ⊑ e' then
23:            sup ← e'
24:        else if a = e ⊒ e' then
25:            sub ← e'
26:        end if
27:    end for
28:    return ⟨eq, sub, sup⟩
29: end function
```

$Pizza \sqsubseteq \bot$, $Pizza \sqsubseteq Eatable$, $Uneatable \equiv \neg Eatable$ (using rule a.2, a.7, a.8, a.11 and a.13, respectively). Without adaptation, instead, only the last axiom would be present in o after concept deletion.

3.3 Rules Correctness Proof

Before stating the proposition about the correctness of the adaptation rules we introduce some definitions and lemmata. For sake of brevity we will interchangeably refer to the axioms and their semantics, according to [4].

Definition 1. *An axiom A_1 **entails** an axiom A_2 iff, for any interpretation I, $I \models A_2 \implies I \models A_1$, that is ${A_2}^I \subseteq {A_1}^I$.*

Definition 2. *An adaptation rule r is **sound** iff $\{LHS(r), prec(r)\}$ entails $RHS(r)$.*

Lemma 1. $\forall C, D, F \in N_{\mathcal{C}} \; . \; C \sqsubseteq D \implies C \sqcup F \sqsubseteq D \sqcup F$.

Proof. By considering the associated semantics the Lemma can be restated as $C^I \subseteq D^I \implies \underbrace{C^I \cup F^I}_{\alpha} \subseteq \underbrace{D^I \cup F^I}_{\beta}$. Assume that the preceding formula does not hold, that is $\alpha \nsubseteq \beta$. This means $\exists x \in \beta \; . \; x \notin \alpha$, and requires that at least one of the following conditions holds:

– $x \in F^I$, but this implies $x \in \alpha$, resulting in a contradiction,
– $x \in C^I$, and thus this implies $C^I \subseteq D^I \implies x \in \alpha$, contradicting the hypothesis. □

Lemma 2. $\forall C, D, F \in N_{\mathcal{C}} \; . \; C \sqsubseteq D \implies C \sqcap F \sqsubseteq D \sqcap F.$

Proof. By considering the associated semantics the Lemma can be restated as $C^I \subseteq D^I \implies \underbrace{C^I \cap F^I}_{\alpha} \subseteq \underbrace{D^I \cap F^I}_{\beta}$. Assume that the preceding formula does not hold, that is $\alpha \not\subseteq \beta$. This means $\exists x \in \beta \; . \; x \notin \alpha$. Note that $x \in \beta$ is equivalent to requiring that $x \in F^I \wedge x \in D^I$ holds. However, $x \in F^I \wedge x \notin \alpha \implies x \notin C^I$. Given that $x \in D^I$ holds, this contradicts the premise $C \sqsubseteq D$. □

Lemma 3. $\forall C, D \in N_{\mathcal{C}} \; . \; C \sqsubseteq D \implies \exists R.C \sqsubseteq \exists R.D.$

Proof. Assume that $\{x \mid \exists y \in C^I \; . \; \langle x, y\rangle \in R^I\} \not\subseteq \{x \mid \exists y \in D^I \; . \; \langle x, y\rangle \in R^I\}$ holds, that is, $\exists R.C \not\sqsubseteq \exists R.D$. This requires that the following condition holds: $\exists \langle x, y\rangle \in R^I \; . \; y \in C^I \wedge y \notin D^I$. But, if such condition holds, then $C \not\sqsubseteq D$, contradicting the premise. □

Lemma 4. $\forall C, D \in N_{\mathcal{C}} \; . \; C \sqsubseteq D \implies \forall R.C \sqsubseteq \forall R.D.$

Proof. Assume that $\{x \mid \forall \langle x, y\rangle \; . \; \langle x, y\rangle \in R^I \implies y \in C^I\} \not\subseteq \{x \mid \forall \langle x, y\rangle \; . \; \langle x, y\rangle \in R^I \implies y \in D^I\}$ holds, that is, $\forall R.C \not\sqsubseteq \forall R.D$. This requires that the following condition holds: $(\exists x \; . \; \forall \langle x, y\rangle \; . \; \langle x, y\rangle \in R^I \implies y \in C^I) \wedge (\exists \bar{y} \; . \; \langle x, \bar{y}\rangle \in R^I \wedge \bar{y} \notin D^I)$. But, if this condition holds, then an $\bar{y}$ exists and R^I is not empty. Therefore, since the left operand of the implication holds, then right operand also does. From this, we obtain $C^I \not\subseteq D^I$, contradicting the premise. □

Proposition 1. *Adaptation rules application preserves ontology satisfiability.*

Proof. Ontology satisfiability is preserved because every adaptation rule is sound. We prove this for each rule separately:

a.1 The proof directly follows from *Concept Equivalence* axiom definition.
a.2 $E \equiv \exists R.C \rightarrow E \sqsubseteq \exists R.D$. $\exists R.C \sqsubseteq \exists R.D$ must hold: thanks to the rule precondition, $C \sqsubseteq D$, we can apply Lemma 3.
a.3 $E \equiv \geq_n R.C \rightarrow E \sqsubseteq \geq_n R.D$. $\geq_n R.C \sqsubseteq \geq_n R.D$ must hold, but it is sufficient that $\{x \mid \exists y \in C^I \; . \; \langle x, y\rangle \in R^I\} \subseteq \{x \mid \exists y \in D^I \; . \; \langle x, y\rangle \in R^I\}$ holds. Thanks to the rule precondition, $C \sqsubseteq D$, we can apply Lemma 3.
a.4 $E \equiv C \sqcup F \rightarrow E \sqsubseteq D \sqcup F$. $C \sqcup F \sqsubseteq D \sqcup F$ holds for Lemma 1 because $C \sqsubseteq D$ holds.
a.5 $E \equiv C \sqcap F \rightarrow E \sqsubseteq D \sqcap F$. $C \sqcap F \sqsubseteq D \sqcap F$ holds for Lemma 2 because $C \sqsubseteq D$ holds.
a.6 $E \equiv \neg C \rightarrow \neg D \sqsubseteq E$. $\neg D \sqsubseteq \neg C$ must hold: the semantics is $\Delta^I \setminus D^I \subseteq \Delta^I \setminus C^I$, but this contradicts $C \sqsubseteq D$.
a.7 $C(a) \rightarrow D(a)$. $C(a) \implies D(a)$ is guaranteed by the rule precondition.
a.8 $E \equiv \forall R.C \rightarrow E \sqsubseteq \forall R.D$. $E \equiv \forall R.C \implies E \sqsubseteq \forall R.D$ holds for Lemma 4 because $C \sqsubseteq D$ holds.

a.9 $E \equiv \leq_n R.C \rightarrow E \sqsubseteq \leq_n R.B$. $\leq_n R.B \sqsubseteq \leq_n R.C$, but it is sufficient that $\{x \mid \exists y \in B^I \,.\, \langle x, y\rangle \in R^I\} \subseteq \{x \mid \exists y \in C^I \,.\, \langle x, y\rangle \in R^I\}$. Thanks to the rule precondition, $B \sqsubseteq C$, we can apply Lemma 3.

a.10 $E \equiv C \sqcup F \rightarrow B \sqcup F \sqsubseteq E$. The proof for $B \sqcup F \sqsubseteq C \sqcup F$ is the dual of the one given in item (a.4).

a.11 $E \equiv C \sqcap F \rightarrow B \sqcap F \sqsubseteq E$. The proof for $B \sqcap F \sqsubseteq C \sqcap F$ is the dual of the one given in item (a.5).

a.12 $E \equiv \neg C \rightarrow E \sqsubseteq \neg B$. the proof for $\neg C \sqsubseteq \neg B$ is the dual of the one given in item (a.6).

a.13 $C \sqsubseteq E \rightarrow B \sqsubseteq E$. The rule precondition, $B \sqsubseteq C$. By transitivity, this implies $B \sqsubseteq E$.

b.1 The proof directly follows from *Role Equivalence* axiom definition.

b.2 $\underbrace{T_0 \circ \ldots \circ T_m \circ R \circ T'_0 \circ \ldots \circ T'_p}_{\alpha} \sqsubseteq T \rightarrow \underbrace{T_0 \circ \ldots \circ T_m \circ Q \circ T'_0 \circ \ldots \circ T'_p}_{\beta} \sqsubseteq T$. assume that $\beta^I \not\subseteq \alpha^I$ holds. This requires that $\exists x_0, \ldots, x_{m+p+3} \,.\, \langle x_0, x_1\rangle \in {T_0}^I \wedge \ldots \wedge \langle x_{m+1}, x_{m+2}\rangle \in Q^I \wedge \langle x_{m+p+2}, x_{m+p+3}\rangle \in {T'_p}^I \wedge \langle x_{m+1}, x_{m+2}\rangle \notin R^I$. This contradicts $Q \sqsubseteq R$.

b.3 $E \equiv \underbrace{\forall R.C}_{\alpha} \rightarrow E \sqsubseteq \underbrace{\forall Q.C}_{\beta}$. Assume that $\alpha^I \not\subseteq \beta^I$ holds. This requires that $\exists x \,.\, x \in \alpha^I \wedge x \notin \beta^I$, that is, $\exists x.((\forall y \,.\, \langle x, y\rangle \in R^I \implies y \in C^I) \wedge (\exists y' \,.\, \langle x, y'\rangle \in Q^I \wedge y' \notin C^I))$. Given that $Q \sqsubseteq R$, if such y' exists, α cannot hold, leading to a contradiction.

b.4 $T \equiv \underbrace{\leq_n R.C}_{\alpha} \rightarrow T \sqsubseteq \underbrace{\leq_n Q.C}_{\beta}$. Assume that $\alpha^I \not\subseteq \beta^I$. This requires that $\exists x \,.\, |\{y \mid y \in C^I \wedge \langle x, y\rangle \in R^I\}| \leq n \wedge |\{y \mid y \in C^I \wedge \langle x, y\rangle \in Q^I\}| > n$. This implies $|Q^I| > |R^I|$, contradicting $Q \sqsubseteq R$.

b.5 $T \equiv R^- \rightarrow Q^- \sqsubseteq T$. Assume that ${Q^-}^I \not\subseteq {R^-}^I$. This requires that $\exists \langle x, y\rangle \,.\, \langle y, x\rangle \in Q^I \wedge \langle y, x\rangle \notin R^I$. This contradicts $Q^I \subseteq R^I$.

b.6 $Disjoint(R, T) \rightarrow Disjoint(Q, T)$. Assume that $R^I \cap T^I = \emptyset \implies Q^I \cap T^I = \emptyset$ does not hold. This requires that $\exists \langle x, y\rangle \in Q^I \wedge \langle x, y\rangle \in T^I \wedge \langle x, y\rangle \notin R^I$ holds, but $\langle x, y\rangle \in Q^I \wedge \langle x, y\rangle \notin R^I$ contradicts $Q \sqsubseteq R$.

b.7 $R(a, b) \rightarrow S(a, b)$. From $R \sqsubseteq S$ we have that $\forall \langle x, y\rangle \,.\, \langle x, y\rangle \in R \implies \langle x, y\rangle \in S$.

b.8 $E \equiv \underbrace{\exists R.C}_{\alpha} \rightarrow E \sqsubseteq \underbrace{\exists S.C}_{\beta}$. Assume that $\alpha^I \not\subseteq \beta^I$. This requires that $\exists x \,.\, \exists y \in C^I \,.\, \langle x, y\rangle \in S^I \wedge \langle x, y\rangle \notin R^I$ holds. This contradicts $R \sqsubseteq S$.

b.9 $E \equiv \underbrace{\exists R.Self}_{\alpha} \rightarrow E \sqsubseteq \underbrace{\exists S.Self}_{\beta}$. Assume that $\alpha^I \not\subseteq \beta^I$. This requires that $\exists x \,.\, \langle x, x\rangle \in S^I \wedge \langle x, x\rangle \notin R^I$ holds. This contradicts $R \sqsubseteq S$.

b.10 $E \equiv \underbrace{\geq_n R.C}_{\alpha} \rightarrow E \sqsubseteq \underbrace{\geq_n S.C}_{\beta}$. Assume that $\alpha^I \not\subseteq \beta^I$. This requires that $\exists x \,.\, |\{y \mid y \in C^I \wedge \langle x, y\rangle \in R^I\}| \geq n \wedge |\{y \mid y \in C^I \wedge \langle x, y\rangle \in S^I\}| < n$. This implies $|R^I| > |S^I|$, contradicting $R \sqsubseteq S$.

b.11 $T \equiv R^- \rightarrow T \sqsubseteq S^-$. Assume that ${R^-}^I \not\subseteq {S^-}^I$. This requires that $\exists \langle x, y \rangle$. $\langle y, x \rangle \in R^I \wedge \langle y, x \rangle \notin S^I$, thus contradicting $R \sqsubseteq S$.

b.12 $T_0 \circ \ldots \circ T_q \sqsubseteq R \rightarrow T_0 \circ \ldots \circ T_q \sqsubseteq S$. This immediately follows, by transitivity, from $R \sqsubseteq S$. □

3.4 Temporal Complexity

Proposition 2. *The time complexity of the algorithm is in $\mathcal{O}(n)$, where n is the number of axioms of the input ontology o.*

Proof. *computePrecond* scans all the axioms of ontology o. For each of them it performs some comparison having a total cost of c_1, so it has a cost of $n \cdot c_1$. The *for* statement of line 4 in Algorithm 1 is executed n times in the worst case (each axiom of the ontology refers to the entity in question). The *for* statement of line 5 is executed $c_2 = |Rules|$ times, where *Rules* is the set of adaptation rules. $satisfies$ test requires a constant (c_3) time for checking the required conditions. Axiom rewriting and its insertion requires constant (c_4) time. The removal of old axioms requires constant time (c_5) too. The overall complexity is therefore equal to $n \cdot c_1 + c_2 \cdot c_3 \cdot c_4 \cdot n + c_5$, that belongs to $\mathcal{O}(n)$. □

4 Experiments

In order to evaluate the practical applicability of our proposal we implemented a Java prototype based on the OWL API library[3]. In OWL API the axioms are immutable objects, and it supports only axiom addition and removal. Whenever possible, the rule application has been simulated with a pair of add and delete changes. In the other cases we employed Java Reflection for directly modifying the involved axiom. In addition to correctness, we also experimentally evaluated the coverage of OWL2 axioms and constructors of our set of rules. The dataset is presented in Table 3 (manual selection on the Web based on ontology size and DL expressivity).

Correctness The developed proof-of-concept prototype has been used for testing correctness of our adaptation rules, the experimental counterpart of the proofs given in Section 3.3. More precisely, the test consists in taking as input a satisfiable ontology composed by the precondition and an axiom corresponding to the LHS of a rule r (modulo alpha renaming of the entity to delete). At this point, using *Hermit* reasoner (v1.3.7)[4], we check the entailment of $RHS(r)[e']$.

Evaluation An entity e is **adaptable** iff it satisfies at least one rule precondition, while an axiom a is **adaptable** iff it at least one rule r s.t. $LHS(r)[e] = a$ exists, in case $prec(r)$ holds w.r.t. e, the axiom is said **fully adaptable**. As an estimation of the practical effectiveness of our algorithm, we consider, for

[3] Available here: `http://owlapi.sourceforge.net/`

[4] Hermit and related information are available at `http://hermit-reasoner.com/`

ID	DL	URI	Axioms	Logical Axioms	(C.1)	(C.2)	(C.3)	(C.4)	(C.2)*	(C.4)*
1.	ALUQ(D)	http://omv.ontoware.org/2009/09/OWLChanges	186	100	100.00	49.49	0.00	N/A	49.49	N/A
2.	SHIN(D)	http://ccdb.ucsd.edu/SAO/1.2	7767	2712	100.00	17.92	97.22	76.17	17.95	76.17
3.	ALEHI+(D)	http://swat.cse.lehigh.edu/onto/univ-bench.owl	243	93	97.67	35.11	36.00	45.16	37.10	45.16
4.	SHOIN(D)	http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine	747	657	94.81	67.88	84.62	66.96	88.29	66.96
5.	ALCO	http://purl.obolibrary.org/obo/ogms.owl	576	84	100.00	50.00	N/A	N/A	50.00	N/A
6.	ALQ(D)	http://owlodm.ontoware.org/OWL2	353	215	90.80	45.50	0.00	N/A	46.91	N/A
7.	SRIQ(D)	http://semanticscience.org/ontology/sio-core.owl	5043	1747	100.00	48.07	100.00	93.55	48.43	93.55
8.	SHOIN	http://www.co-ode.org/ontologies/pizza/pizza.owl	939	712	100.00	22.14	75.00	100.00	22.15	100.00
9.	SHOIN(D)	http://sweet.jpl.nasa.gov/2.1/reprSciUnits.owl	503	475	100.00	8.20	33.33	0.00	42.86	0.00
10.	SR	http://purl.obolibrary.org/obo/hao/2011-11-03/hao.owl	20027	8493	96.22	38.53	0.00	N/A	50.47	N/A
11.	SROIF	http://purl.obolibrary.org/obo/ido.owl	3499	1025	100.00	34.67	86.67	67.02	36.92	67.73
12.	SROIF	http://ontology.neuinfo.org/NIF/Dysfunction/NIF-Dysfunction.owl	5649	350	100.00	50.00	N/A	N/A	50.00	N/A
13.	SHIN(D)	http://aims.fao.org/aos/geopolitical.owl	23527	22834	100.00	100.00	100.00	100.00	100.00	100.00
14.	S	http://human.owl	30364	11545	99.82	49.84	0.00	N/A	49.84	N/A
15.	ALE	http://mouse.owl	11043	4838	67.02	36.99	0.00	N/A	38.93	N/A
16.	ALCHOIN	http://owl.man.ac.uk/2005/07/sssw/people.owl	396	108	100.00	69.94	71.43	81.25	71.18	81.25
17.	SROIF	http://ontology.neuinfo.org/NIF/BiomaterialEntities/NIF-GrossAnatomy.owl	19849	2930	99.75	39.62	18.18	100.00	39.62	100.00
18.	SROIN(D)	http://purl.obolibrary.org/obo/flu/dev/flu.owl	874	204	100.00	100.00	100.00	100.00	100.00	100.00
19.	ALCOIF	http://www.owl-ontologies.com/generations.owl	60	38	94.44	77.14	75.00	100.00	100.00	100.00
20.	AL(D)	http://protege.stanford.edu/plugins/owl/owl-library/ka.owl	404	216	100.00	57.03	0.00	N/A	57.18	N/A
21.	SROIF	http://ontology.neuinfo.org/NIF/BiomaterialEntities/NIF-Cell.owl	3508	398	100.00	46.73	0.00	N/A	46.73	N/A
22.	SOIN(D)	http://www.owl-ontologies.com/travel.owl	145	93	100.00	49.61	33.33	100.00	50.00	100.00
23.	ALUHN	http://www.hozo.jp/owl/YAMATO20120714.owl	4428	2444	100.00	51.69	95.63	11.67	51.69	11.67
24.	ALHIF(D)	http://purl.org/net/ontology/beer.owl	165	81	84.48	42.86	22.22	100.00	42.86	100.00
25.	SH(D)	http://msi-ontology.sourceforge.net/ontology/NMR.owl	1096	290	100.00	50.00	N/A	N/A	50.00	N/A
26.	SHIF(D)	http://www.nada.kth.se/~mehrana/Delegation.owl	103	63	63.16	40.91	80.00	81.25	40.91	81.25
27.	ALCRIQ(D)	http://www.biomodels.net/kisao/KISAO	2044	647	100.00	42.70	100.00	92.23	42.82	92.47
28.	ALCO	http://purl.obolibrary.org/obo/omrse.owl	99	25	100.00	50.00	0.00	N/A	50.00	N/A

Table 3. Dataset and coverage results presented in Section 4. Ontologies 14 and 15 are part of the dataset used by *Ontology Alignment Evaluation Initiative*, and are available at http://oaei.ontologymatching.org/. Coverage results are unaggregated and based on the data of Table 4. N/A means the ontology contains no roles/adaptable roles.

each ontology in our dataset, the following scenario: we simulate the deletion of each single entity, in isolation, and we take into account the percentage of adaptable ones (i.e., such that another entity suitable for reformulation exists). For each of these adaptable entities, we also inspect how many axioms involving them would be adapted instead of simply deleted. For this reference scenario we defined *Coverage* measure as: **(C.1)** the percentage of adaptable concepts (resp. roles **(C.3)**) out of the total number of concepts (resp. roles), and **(C.2)** the percentage of adaptable axioms w.r.t. the deleted concept (resp. role, **(C.4)**) out of the number of axioms to be deleted (that is, presenting the deleted entity in their signature). The (C._)* variants count the fully adaptable axioms, and evaluate the completeness of our adaptation rules (the complement of the fully adaptable axioms is not supported by our rules).

In Table 3 the coverage for each ontology in isolation is reported (computed from the raw data of Table 4), while the result considering the dataset as a whole ontology is the following: (C.1) 93.247%, (C.2) 41.757%, (C.2*) 44.185%, (C.3) 73.647%, (C.4) 79.63%, (C.4*) 80.847%. Table 3 shows that 10 out of 12 of the worst performing ontologies w.r.t. role coverage ((C.3), (C.4) and (C.4)*) are expressed in a DL missing role hierarchy constructs (identified by letter H in the DL name). Without role hierarchy constructs only role equality can be used for adaptation, thus reducing the number of adaptable roles. Concept coverage (C.1) presents, instead, high values (above 60%) for all the considered ontologies, independently from the DL they are expressed with. This is not surprising because concept hierarchy constructs are available for DLs at least as expressive as $\mathcal{AL}$. On the contrary, coverage results for concept rules w.r.t. OWL2 axioms and constructors seem to be unrelated to either the underpinning DL or the ontology size (in terms of number of axioms and/or entities). For instance, the ontologies with worst values for (C.2)* are 2. ($\mathcal{SHIN}(\mathcal{D})$), 8. ($\mathcal{SHOIN}(\mathcal{D})$) 11. ($\mathcal{SROIF}$) 3. ($\mathcal{ALEHI}+(\mathcal{D})$) and 15. ($\mathcal{ALE}$), with very different number of concepts and axioms (Table 4). Similarly, among the best results for (C.2)* the expressivity ranges from $\mathcal{AL}(\mathcal{D})$ to $\mathcal{SROIN}(\mathcal{D})$, again with varying number of axioms and concepts. Ideally the proposal should adapt all the axioms: (C.2)*, in particular, is far from this result, but it is well known that OWL2, despite being based on $\mathcal{SROIQ}$, adds new constructors and axioms, that are derivable from $\mathcal{SROIQ}$ ones (they do not add expressive power). For example, *Concept Disjointness* axiom (*i.e.,* $Disjoint(C, D)$, with $C, D \in \mathcal{N}_C$) is only a shortcut for $C \sqcap D \sqsubseteq \bot$[5]. Our prototype strictly applies the rules of Table 1 and Table 2, so it cannot directly process the axioms and constructors not available in $\mathcal{SROIQ}$ DL, thus diminishing the number of adaptable axioms.

5 Future Work

The paper represents, to the best of our knowledge, the first proposal for ontology adaptation upon updates. In addition, the algorithm is totally automatic and supports ontology expressivity up to $\mathcal{SROIQ}$, on top of which OWL2 is defined.

[5] Refer to [9], Chapter 9, for further examples and details.

The present paper could be extended in several directions. The set of adaptation rules is a preliminary proposal, we plan to further enrich it in order to increase the coverage rate reported in Section 4 and to consider reasonable alternatives for each single rule (*e.g.*, sound alternatives for a.8 could be $C \sqsubseteq D, E \equiv \forall R.C \rightarrow \forall R.D \sqsubseteq E$ or $B_0 \ldots B_n \sqsubseteq C, E \equiv \forall R.C \rightarrow E \sqsubseteq \forall R. \bigsqcup_{i=0}^{n} B$). We also plan to consider the integration of anonymous entities (*e.g.*, using $\top$ as superclass). Another possible extension is the integration of a complex update (*e.g.*, concept merge and split) proposals, such as [2]. The relationship between DL updates and *Belief Revision* has been investigated [8], we plan to further investigate it w.r.t. our proposal. We also intend to improve our prototype up to a full support of OWL2. Our final goal will be a *Protégé* plugin, from which we hope to receive feedbacks from the community of ontology engineers and practitioners. The experimental evaluation will also be strengthened with an extended ontology dataset and temporal profiling of the prototype.

References

1. Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., Antoniou, G.: Ontology change: Classification and survey. Knowl. Eng. Rev. 23(2), 117–152 (2008)
2. Hartung, M., Groß, A., Rahm, E.: COnto–Diff: Generation of Complex Evolution Mappings for Life Science Ontologies. Journal of Biomedical Informatics (2012)
3. Hartung, M., Groß, A., Rahm, E.: Rule-based Generation of Diff Evolution Mappings between Ontology Versions. CoRR abs/1010.0122 (2010)
4. Horrocks, I., Kutz, O., Sattler, U.: The Even More Irresistible SROIQ. In: Principles of Knowledge Representation and Reasoning – KR 2006. pp. 57–67 (2006)
5. Katsuno, H., Mendelzon, A.O.: Propositional Knowledge Base Revision and Minimal Change. Artificial Intelligence 52(3), 263–294 (1991)
6. Kondylakis, H., Plexousakis, D.: Ontology Evolution: Assisting Query Migration. Conceptual Modeling – ER 2012 pp. 331–344 (2012)
7. Noy, N., Klein, M.: Ontology Evolution: Not the same as Schema Evolution. Knowledge and Information Systems 6(4), 428–440 (2004)
8. Ribeiro, M.M., Wassermann, R., Antoniou, G., Flouris, G., Pan, J.: Belief Contraction in Web-Ontology Languages. In: Workshop on Ontology Dynamics, IWOD (2009)
9. Rudolph, S.: Foundations of Description Logics. Reasoning Web. Semantic Technologies for the Web of Data pp. 76–136 (2011)
10. Stojanovic, L.: Methods and Tools for Ontology Evolution. Ph.D. thesis, University of Karlsruhe (2004)
11. W3C as Hitzler, P. and Krötzsch, M. and Parsia, B. Patel-Schneider, P.F. and Rudolph, S.: OWL 2 Web Ontology Language Primer. `http://www.w3.org/TR/owl2-primer/` (2009)

ID	Concepts	Adaptable Concepts	Concept Axioms	Adaptable Concept Axioms	Roles	Adaptable Roles	Role Axioms	Adaptable Role Axioms	Unsatisfiable Concept Axioms	Unsatisfiable Role Axioms
1.	100	100	198	98	2	0	0	0	0	0
2.	736	736	5129	919	36	35	256	195	8	0
3.	43	42	131	46	25	9	31	14	7	0
4.	77	73	411	279	13	11	345	231	95	0
5.	92	92	168	84	0	0	0	0	0	0
6.	87	79	367	167	44	0	0	0	11	0
7.	1021	1021	2823	1357	184	184	946	885	21	0
8.	100	100	1477	327	8	6	180	180	1	0
9.	12	12	183	15	6	2	137	0	148	136
10.	1930	1857	9809	3779	4	0	0	0	2321	0
11.	509	509	2189	759	30	26	285	191	133	3
12.	352	352	700	350	0	0	0	0	0	0
13.	12	12	460	460	6	6	4888	4888	0	0
14.	3304	3298	10880	5423	2	0	0	0	0	0
15.	2744	1839	6256	2314	3	0	0	0	312	0
16.	60	60	173	121	14	10	48	39	3	0
17.	1628	1624	6116	2423	11	2	86	86	0	0
18.	213	213	447	447	19	19	83	83	0	0
19.	18	17	35	27	4	3	20	20	8	0
20.	96	96	377	215	60	0	0	0	1	0
21.	373	373	794	371	2	0	0	0	0	0
22.	35	35	129	64	6	2	11	11	1	0
23.	925	925	4589	2372	183	175	1542	180	0	0
24.	58	49	105	45	9	2	6	6	0	0
25.	301	301	580	290	0	0	0	0	0	0
26.	19	12	22	9	20	16	48	39	0	0
27.	202	202	1335	570	9	9	386	356	4	1
28.	27	27	50	25	2	0	0	0	0	0

Table 4. Raw data used for coverage analysis of Section 4. Unsatisfied axioms stands for axioms matching the LHS of a rule having a precondition not satisfied by the entity under deletion.

Checking and Repairing Ontological Naming Patterns using ORE and PatOMat

Ondřej Zamazal[1], Lorenz Bühmann[2], and Vojtěch Svátek[1]

[1] Knowledge Engineering Group, University of Economics Prague, Czech Republic
{ondrej.zamazal}|svatek@vse.cz
[2] AKSW research group, University of Leipzig, Germany,
buehmann@informatik.uni-leipzig.de

Abstract. Analysis of the naming of entities across ontological structures can help reveal both naming issues and underlying conceptualization issues. Cross-entity naming analysis thus extends the standard logical satisfiability checking by an extra, less rigorous and reliable but often farther reaching layer. We show how such naming patterns can be applied within the transformation pattern paradigm used by the *PatOMat* transformation framework. We describe how the PatOMat tool has been integrated into the (logic-oriented) *Ontology Repair and Enrichment* tool (ORE), and present the results of application of a prominent naming pattern, 'non-matching child', on a collection of linked data vocabularies.

Keywords: Naming pattern, Ontology Repair, PatOMat, ORE

1 Introduction

During the decades of knowledge engineering research, there has been recurrent dispute on how the natural language structure influences the structure of formal knowledge bases and vice versa. A large part of the community seems to recognise that the content expressed in formal representation languages, such as the semantic web ones, should be accessible not only to logical reasoning machines but also to humans and NLP procedures, and thus resemble the natural language as much as possible.[3] We build upon the assumption that naming in ontologies matters, can be sensibly designed, and to some degree even automatically identified in existing ontologies, with the help of *naming patterns*. Current ontology debugging methods, mostly dealing with the logical structure of the ontology only, can thus be extended by debugging of naming issues. Detecting improper or awkward naming should be ideally followed by repairing suggestions. While in the biomedical field there have already been efforts in naming analysis, e.g., in [2, 9], naming in the broad field of linked data vocabularies (where domain-specific heuristics cannot be applied) has rarely been addressed. Generic tools such as OntoCheck [8] have so far been only equipped with very simple tests such as that of name length or presence of a concrete sub-token (such as 'and'). The

[3] See for example the arguments by Y. Wilks in [7]

presented approach thus contributes to filling in a missing piece: domain-neutral cross-entity analysis.

The paper follows up on earlier research described in [11]. In contrast to [11], where the analysis of presence of the 'non-matching child' pattern was carried out manually and only qualitative results (for three ontologies) were presented, we now

- carry out the pattern detection fully *automatically*, by means of a versatile ontology transformation framework, *PatOMat* [12], with declaratively represented patterns
- in this context, also consider a simple form of pattern-based *repair* of the discovered issue
- provide a lightweight integration of naming analysis and logical satisfiability analysis within the *Ontology Repair and Enrichment* tool (ORE) [5]
- present the empirical results of analysis on a larger number of linked data vocabularies included in the respected *Linked Open Vocabularies* collection.

The paper is structured as follows. Section 2 briefly surveys the PatOMat transformation framework. Section 3 explains the NMC (non-matching child) pattern and describes its implementation via PatOMat transformation patterns. Section 4 describes the integration of the whole functionality into ORE. Section 5 reports on an experiment in NMC pattern detection over 16 ontologies (namely, popular linked data vocabularies). Section 6 then wraps up the paper.

2 PatOMat Framework and Naming Patterns

PatOMat framework principles The *PatOMat* Framework[4] has been originally designed with the goal of transforming ontologies between 'structural' modelling styles, e.g., via de/reifying properties, metamodelling classes by individuals, switching between object and data properties, and the like. However, the entity naming aspect has been considered from the beginning.

The central notion in PatOMat is that of transformation pattern (TP). A TP contains two *ontology patterns* (source OP and target OP) and the description of the transformation betweem them, called *pattern transformation* (PT). For instance, we can specify a very simple TP such that a subsumption relation between two classes (as source, OP1) should be transformed to a SKOS[5] taxonomic relationship between two individuals (as target, OP2). A schematic description follows.[6]

```
OP1: Class: ?OP1_A subClassOf ?OP1_B
OP2: Class: ?OP2_A skos:broader ?OP2_B
PT:  ?OP1_A = ?OP2_A    ?OP1_B = ?OP2_B.
```

[4] [12] provides more details about the framework, and at `http://owl.vse.cz:8080/tutorial/` there is a fully-fledged tutorial for the current version.

[5] `http://www.w3.org/TR/skos-primer/`

[6] OP1 and OP2 contain axioms in frame-based variant of Manchester syntax, `http://www.w3.org/TR/owl2-manchester-syntax/`

The representation of OPs is based on the OWL 2 DL profile. However, while an OWL ontology refers to particular entities, e.g. to class `Person`, in the patterns we generally use *placeholders*, e.g. $?OP1\_A$. Entities are specified (i.e. placeholders are instantiated) at the time of instantiation of a pattern. An OP consists of *entity declarations* (referring to placeholders or concrete entities), *axioms* and *naming detection patterns* (NDPs); the last capture the naming aspect of the OP important for its detection, see below. A PT consists of a set of *transformation links* and a set of *naming transformation patterns* (NTPs). Transformation links are either *logical equivalence relationships* or *extralogical relationships* holding between pairs of entities of different type (such as class vs. individual, as in our example above). NTPs serve for generating new names for original or newly created entities.

PatOMat currently supports naming operations at the level of short URIs; its extension to *rdfs:label* values would however be straightforward. Various token separators, such as underscore, hyphen or camel-case, are supported.

PatOMat implementation The framework prototype implementation is available either as a *Java library* or as three *core services*.[7] The whole transformation is divided into three steps, which correspond to the three services:

- The *OntologyPatternDetection* service takes the transformation pattern and a particular original ontology on input, and returns the binding of entity placeholders on output, in XML. The structural/logical aspect is captured in the structure of an automatically generated SPARQL query;[8] the naming aspect is dealt with based on its description within the source pattern.
- The *InstructionGenerator* service takes the particular binding of placeholders and the transformation pattern on input, and returns particular transformation instructions on output, also in XML. Transformation instructions are generated according to the transformation pattern and the pattern instance.
- The *OntologyTransformation* service takes the particular transformation instructions and the particular original ontology on input, and returns the transformed ontology on output. The service is based on our specific implementation over OWL-API,[9] and enables operations on *axioms*, *entities* and adding *OWL annotations*.

The process of transformation is decomposed into parts in order to enable an user intervention within the whole workflow. User intervention can be carried out using a generic graphical tool, *GUIPOT* [13].

[7] All accessible via the web interface at `http://owl.vse.cz:8080/`.
[8] `http://www.w3.org/TR/rdf-sparql-query/`
[9] `http://owlapi.sourceforge.net/`

3 Non-Matching Child (NMC) Pattern

It is quite common in ontologies that a subclass has the *same head noun* as its parent class.[10] By an earlier study [11] we estimate that in ontologies for technical domains this simple pattern is verified in 50–80% of class-subclass pairs such that the subclass name is a *multi-token* one. This number further increases if we consider *thesaurus correspondence* (synonymy and hypernymy) rather than literal string equality. In fact, the set-theoretic nature of taxonomic path entails that the correspondence of head nouns along this path should be close to 100% in principle; the only completely innocent deviations from it should be those caused by incomplete thesauri. In other words, any violation of head noun correspondence may potentially indicate a (smaller or greater) problem in the ontology. Prototypical situations are:

- Inadequate use of class-subclass relationship, typically in the place of whole-part or class-instance relationship, i.e., a *conceptualisation error* frequently occurring in novice ontologies.
- *Name shorthanding*, typically manifested by use of adjective, such as 'State-Owned' (subclass of 'Company').

While the former requires complex refactoring of the ontology fragment, the latter can be healed by propagation of the parent name down to the child name.

NMC pattern in PatOMat Let us now show how to capture the NMC pattern within a PatOMat TP. The source OP is here just a *subClassOf* relationship between two classes. There are however two variants of this source OP: one using *subClassOf* and one using *directSubClassOf*, the latter operating on class pairs identified by a reasoner.[11]

An NDP within the source OP can consist of several naming operations such as detection of a head noun; their results can then be compared using different methods. We designed two variants of such an NDP:

1. comparing whether ?$OP1\_A$ has the same head noun as ?$OP1\_P$ (e-variant, for 'equality') or
2. comparing whether head noun of ?$OP1\_P$ is a hypernym of head noun of ?$OP1\_A$ (t-variant, for 'thesaurus').

The target OP has only one variant, which is structurewise identical to the source OP, i.e., `Class: ?OP2_A SubClassOf: ?OP2_P` (and there is no NDP part). Finally, the PT contains transformation links specifying equality of ?$OP1\_A$ and ?$OP2\_A$ and analogously for ?$OP1\_P$ and ?$OP2\_P$. More interestingly, it also includes an NTP, which represents the naming *repair* step. It specifies that ?OP_A should be extended by the head noun of ?$OP1\_P$.

In combination, we can have four variants of the NMC pattern, of which we consider three: (1) Se, St and Dt.[12] The Se variant simply matches the head

[10] The head noun is typically the last token, but not always, in particular due to possible prepositional constructions, as, e.g., in 'HeadOfDepartment'.

[11] We used Pellet, `http://pellet.owldl.com/`.

[12] All variants available at `http://nb.vse.cz/~svabo/patomat/tp/np/`

nouns, the *Dt* variant needs to employ a reasoner, and both *t* variants employ a thesaurus in order to verify the hypernym relationships. In our case we use *WordNet* [6]. In order to traverse hypernym relations we first get all senses of a given word and retrieve hypernyms of all senses. Then we check whether the lemma of a given word is the same as the lemma of one of hypernyms. If it is not the case, it continues to the next level of hypernyms. By experience, we set up the number of levels to five.

In the experiment described in Section 5, we used the *St* variant of the pattern, but, referring to the conjecture formulated at the beginning of this section, distinguished between single- and multi-token child.

4 Integration into ORE

The ORE[13] (Ontology Repair and Enrichment) tool was designed so as to allow knowledge engineers to improve knowledge bases in the form of SPARQL endpoints and OWL ontologies. ORE helps fixing several kinds of problems such as logical errors, i.e., unsatisfiable classes and inconsistencies, by applying state-of-the-art methods [3, 4], and, newly, the naming problems described in this paper. Additionally, ORE allows for the semi-automatic enrichment of knowledge base schemas by suggesting OWL axioms generated by the application of machine learning algorithms [1] on the underlying instance data. These data adhering axioms, if accepted by the user, can result in a more expressive ontology, which can for example enable more powerful querying.

The PatOMat framework is integrated into ORE by means of a separate task and visualized in a single view as shown in Figure 1. Here the user can select a naming pattern, as for example `non-matching child1` (corresponding to the *St* variant of the non-matching child pattern), in the leftmost list (①). PatOMat then detects instances of the selected pattern in the currently loaded ontology, e.g., `[?OP1_P=Contribution;?OP1_A=Poster]` (see ②). For the selected pattern instances the user will be provided a list of renaming instructions (see ③), for example to rename the class `Poster` to `PosterContribution`, which can then be used to transform the ontology and solve the detected naming issues.

5 NMC Pattern Detection Experiment

We carried out a small experiment on 16 randomly selected vocabularies from the Linked Open Vocabularies (LOV) catalog.[14] They belong to four of the eight major clusters suggested by the LOV curators: City (related to personal, social and governmental data), Library, Media and Market. Four of the vocabularies, *ontopic*, *swc*, *wi* and *gc*, imported other vocabularies; we considered them together with these imports. For completeness we also include two vocabularies that did not contain any subclass axioms.

[13] `http://aksw.org/Projects/ORE.html`
[14] `http://lov.okfn.org`

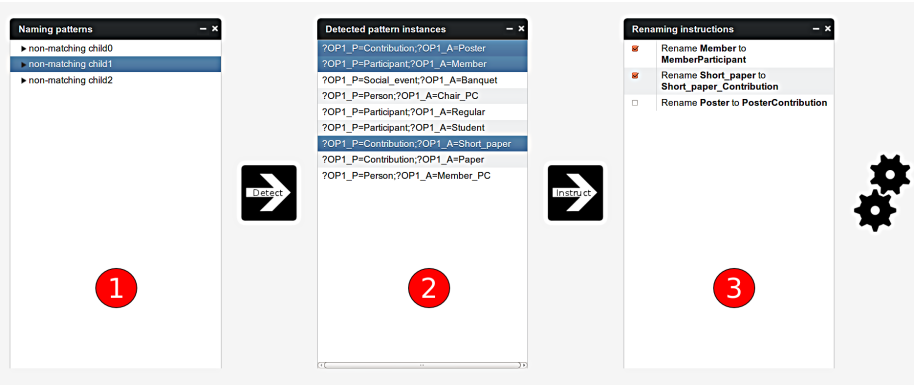

Fig. 1. Screenshot of the PatOMat view in the ORE tool.

The summary of the results is in Table 1. The first two columns display the nickname of the vocabulary within LOV,[15] its catalogued name and cluster. The next three columns show the total number of asserted subclass axioms, the number of axioms matching the NMC pattern (the ordering in the table is in the descending order of this field), and the respective ratio. The following three columns are analogous, but only refer to subclass axioms where the subclass has a multi-token name. The last two columns show the time spent (in seconds) and the average time corresponding to one subclass axiom.

The proportion of axioms satisfying the NMC pattern ('Ratio all' column) ranges from 0% to 87%. The results however seem to confirm our conjecture that subclasses with *multi-token* names are more likely to follow (at least via thesaurus correspondence) the head noun of their parent class name. For 11 of the 16 vocabularies, the frequency of occurrence of the NMC pattern was reduced by focusing on multi-token subclasses, and only for 2 (*swc* and *gr*) it increased.

Manual analysis of the 'alerts' revealed several interesting cases:

- Although possibly with some 'philosophical' excuse, the class-subclass pair 'Topic'–'TopicSignature' in *ontopic* is suspect for tacit *partonomy*, as the latter is commented as 'the subcollection of terms populating a Topic' (a set of terms is likely not same as the topic it populates).
- Head noun of the superclass is sometimes a *meta-level* term. For example, in *swc* the class 'ProgrammeCommitteeMember' is a subclass of 'Role'; obviously, if this subclass is populated by 'Person' entities, they would become instances of 'Role', which would be undesirable. In *gnd*, which is a taxonomy of keyword types rather than a true data vocabulary, class 'PlaceOrGeographicName' has subclasses such as 'MemberState'; however, the latter could be, in the linked data setting, populated by true 'state' entities (rather than just by their names viewed as keywords).

[15] The URI of the catalog item page is `http://lov.okfn.org/dataset/lov/details/vocabulary_<nickname>.html`.

Nick name	Name/topic	Cluster	Subcl all	Patt all	Ratio all	Subcl MT	Patt MT	Ratio MT	Time total	Time avg.
ontopic	Ontopic	Library	182	91	50%	95	39	41%	33	0.18
swc	SemWeb Conference	City	100	36	36%	55	25	45%	17	0.17
gnd	GND	Library	47	31	66%	37	23	62%	17	0.36
mvco	Media Value Chain	Media	32	24	75%	13	6	46%	11	0.34
pattern	Pattern	Library	15	13	87%	0	0	-	9	0.60
gr	GR - GoodRelations	Market	19	10	53%	13	8	62%	12	0.63
wi	Weighted Interests	City	20	9	45%	8	2	25%	24	1.20
bibo	Bibliographic	Library	53	8	15%	15	2	13%	11	0.21
frbr	Core FRBR Concepts	Library	27	7	26%	6	0	0%	11	0.41
gc	oeGOV Governm.Core	City	11	6	55%	3	1	33%	14	1.27
foaf	FOAF	City	10	3	30%	4	0	0%	9	0.90
sioc	SIOC	City	6	2	33%	1	0	0%	9	1.50
pna	Press.net Asset	Media	3	2	67%	0	0	0%	8	2.67
chord	OMRAS2 Chord	Media	3	1	33%	2	0	0%	9	3.00
comm	Incident communication	City	0	0	-	0	0	-	8	-
part	Participation Schema	City	0	0	-	0	0	-	7	-
Total			528	243	46%	252	106	42%		

Table 1. Results of vocabulary analysis wrt. the NMC pattern

- The previous is a special case of *name shorthanding*. A more typical case is such that no head *noun* can be detected at all, e.g., with pair such as 'PoliticalSystem'–'Tribal' (in *gc*) or 'Publication'–'Unpublished' (in *swc*).
- Some vocabularies redefine common terms in their *specific manner*, which generates false alerts. For example, in *bibo*, 'CourtReporter' is subclass of 'Periodical' (while a 'reporter' would not primarily be viewed as periodical) and 'LegalCaseDecision' is a subclass of 'LegalCaseDocument' (while a 'decision', in the general sense, is not a document). Similarly, *ontopic* has 'TimeInterval' as subclass of 'Region' (here the unusual choice of term 'region' follows from the upper-level nature of the ontology).
- Sometimes the *head noun detection* fails due to non-intuitive agglutination of tokens. A grammatically sound one is 'SubjectHeadingSensoStricto' (in *gnd*), unmatched to its parent 'SubjectHeading'. Less sound seem to be, e.g., 'PaymentMethodCreditCard or 'QuantitativeValueFloat' (in *gr*).
- An unusual *case* setting may make the *tokenizer* fail and thus lead to a false alert, e.g., for a class named 'Vevent' (a kind of 'event' in *swc*).

The relatively high computation times are partly owing to SPARQL query evaluation and partly to WordNet traversal. However, given the offline nature of the task, they are not prohibitive.

6 Conclusions and Future Work

Many ontologies, including linked data vocabularies, have recently been created, often with little concern for naming coherence, and sometimes even with conceptualization flaws (also reflected by naming incoherence). Visual analysis of the naming aspect of their taxonomic structures is typically feasible, as most ontologies/vocabularies are not extremely large. However, allowing the user to focus on 'suspicious' cases (and ignoring those apparently sound) can be helpful.

We present a solution for such machine-supported analysis, which combines the functionality of an ontology debugging (and enrichment) tool, ORE, with that of a pattern-based ontology transformation framework, PatOMat, and with online access to WordNet. Empirical results of naming analysis (regarding the NMR pattern) on 16 linked data vocabularies have been presented.

In future we plan to come up with more sophisticated naming/transformation patterns (e.g., indicating a taxonomy/partonomy mismatchh), and involve a larger number of vocabularies in the experiment.

The research is supported by the EU ICT FP7 under No.257943, LOD2 project.

References

1. Bühmann L., Lehmann J.: Universal OWL Axiom Enrichment for Large Knowledge Bases. In: EKAW 2012, Galway, Ireland, 2012.
2. Fernandez-Breis, J. T., Iannone, L., Palmisano, I., Rector, A. L., Stevens, R.: Enriching the Gene Ontology via the Dissection of Labels Using the Ontology Pre-processor Language. EKAW 2010: 59-73.
3. Horridge M., Parsia B., Sattler U.: Laconic and Precise Justifications in OWL. In: ISWC 2008, Karlsruhe, Germany, 2008.
4. Kalyanpur A., Parsia B., Horridge M., Sirin E.: Finding all justifications of OWL DL entailments. In: ISWC 2007, 2007.
5. Lehmann J., Bühmann L.: ORE - a tool for repairing and enriching knowledge bases. In: ISWC'10, Shanghai, China, 2010.
6. Miller G. A. WordNet: A Lexical Database for English. *CACM*, 1995.
7. Nirenburg S., Wilks Y.: Whats in a symbol: Ontology and the surface of language. *Journal of Experimental and Theoretical AI*, 2001.
8. Schober, D., Tudose, I., Svátek, V., Boeker, M.: OntoCheck: verifying ontology naming conventions and metadata completeness in Protégé 4. *J. Biomed. Semantics*, 2012, 3(Suppl 2):S4.
9. Schober, D., Smith, B., Lewis, S. E., Kusnierczyk, W., Lomax, J., Mungall, C., Taylor, C. F., Rocca-Serra, P., Sansone, S.-A.: Survey-based naming conventions for use in OBO Foundry ontology development. *BMC Bioinformatics* 10 (2009).
10. Svátek V., Šváb-Zamazal O., Presutti V.: Ontology Naming Pattern Sauce for (Human and Computer) Gourmets. In: Workshop on Ontology Patterns at ISWC09.
11. Šváb-Zamazal O., Svátek V.: Analysing Ontological Structures through Name Pattern Tracking. In: EKAW-2008, Acitrezza, Italy, 2008.
12. Šváb-Zamazal O., Svátek V., Iannone L.: Pattern-Based Ontology Transformation Service Exploiting OPPL and OWL-API. In: EKAW 2010.
13. Zamazal O., Dudáš M., Svátek V.: User-Friedly Pattern-Based Transformation of OWL Ontologies. In: EKAW 2012, Galway, Ireland, 2012.