OntoComP System Description

Barış Sertkaya *

Theoretical Computer Science, TU Dresden, Germany
sertkaya@tcs.inf.tu-dresden.de

Abstract. We describe OntoComP, a Protégé 4 plugin that supports knowledge engineers in completing DL-based ontologies. More precisely, OntoComP supports a knowledge engineer in checking whether an ontology contains all the relevant information about the application domain, and in extending the ontology appropriately if this is not the case. It acquires complete knowledge about the application domain efficiently by asking successive questions to the knowledge engineer. By using novel techniques from Formal Concept Analysis, it ensures that, on the one hand, the interaction with the knowledge engineer is kept to a minimum, and, on the other hand, the resulting ontology is complete in a certain well-defined sense.

1 Introduction

Since the standardization of OWL as the ontology language for the Semantic Web [9], several ontology editors now support OWL [12, 11], and ontologies written in OWL are employed in more and more applications in various domains. As the number and size of these ontologies grow, tools that support improving and maintaining their quality become more important. The tools available until now mostly deal with detecting inconsistencies and inferring consequences, i.e., implicit knowledge that can be deduced from the knowledge explicitly represented in the ontology. There are also promising approaches that allow to pinpoint the reasons for inconsistencies and for certain unwanted consequences. These approaches address the quality dimension of *soundness* of an ontology, both within itself (consistency) and w.r.t. the intended application domain (no unwanted consequences). In our previous work [2, 15], we have considered a different quality dimension, namely *completeness* of the knowledge in an ontology. We have provided a formally well-founded technique called *ontology completion*, that supports the ontology engineer in checking whether an ontology contains all the relevant information about the application domain, and in extending the ontology appropriately if this is not the case. In ontology completion, given an application domain and a DL-based ontology describing it, we are interested in checking whether all relevant constraints that hold between concepts in the domain are captured by the TBox, and whether all relevant individuals existing in the domain are represented in the ABox.

* Supported by German Research Foundation (DFG) under grant BA 1122/12-1

Clearly, these questions cannot be answered by an automated tool alone. In order to check whether a given relationship between concepts—which does not already follow from the TBox—holds in the domain, one needs to ask a domain expert, and the same is true for questions regarding the existence of individuals not described in the ABox. The role of the ontology completion tool here is to ensure that the expert is asked as few questions as possible; in particular, she should not be asked trivial questions, i.e., questions that could actually be answered based on the represented knowledge.

2 Motivating example

As an example of how ontology completion supports the ontology engineer in practice, consider the OWL ontology for human protein phosphatases that has been described and used in [21]. This ontology was developed based on information from peer-reviewed publications. The human protein phosphatase family has been well characterised experimentally, and detailed knowledge about different classes of such proteins is available. This knowledge is represented in the terminological part of the ontology. Moreover, a large set of human phosphatases has been identified and documented by expert biologists. These are described as individuals in the assertional part of the ontology. One can now ask whether the information about protein phosphatases contained in this ontology is complete. That is, are all the relationships that hold among the introduced classes of phosphatases captured by the constraints in the TBox, or are there relationships that hold in the domain, but do not follow from the TBox? Are all possible kinds of human protein phosphatases represented by individuals in the ABox, or are there phosphatases that have not yet been included in the ontology or even not yet been identified?

Clearly, these questions need to be answered by a biologist. In this example, answering a non-trivial question regarding human protein phosphatases may require the biologist to study the relevant literature, query existing protein databases, or even to carry out new experiments. Thus, the expert may be prompted to acquire new biological knowledge.

3 Ontology Completion

The key technologies lying under ontology completion are Description Logic reasoning, and the attribute exploration method developed in Formal Concept Analysis (FCA) [7]. FCA is a field of applied mathematics that aims to formalize the notions of a concept and a conceptual hierarchy by means of mathematical tools. It is used for conceptual data analysis and knowledge processing. Attribute exploration is a knowledge acquisition method of FCA that is used to acquire complete knowledge about an application domain by asking successive questions to a domain expert. In [2, 15] we have presented an adaptation of this method for completing a DL knowledge base w.r.t. a fixed model. It assumes the existence of a domain expert that has (or can obtain) enough information about an

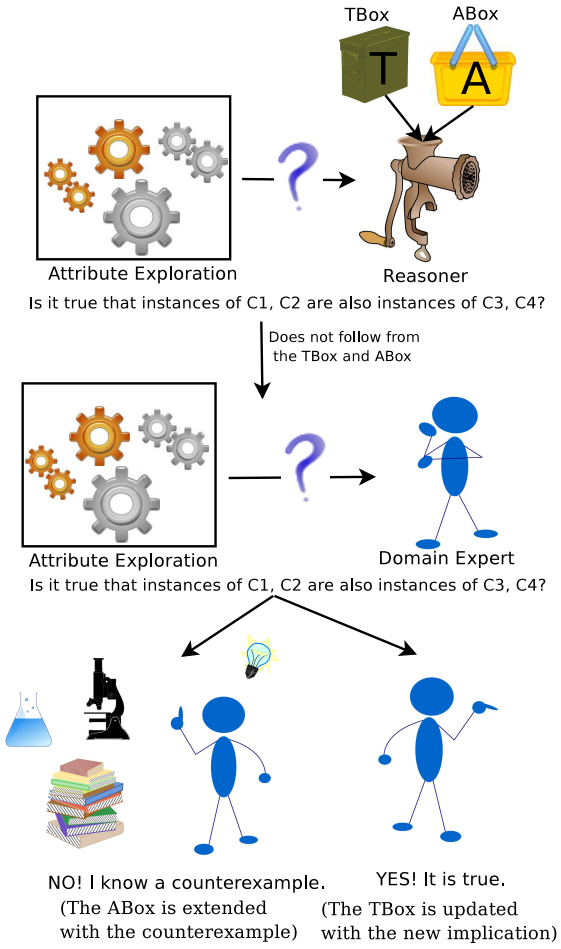

Fig. 1. Ontology completion process

application domain $\mathcal{I}$ to be able to answer implication questions of the form "*Is* $L \to R$ *refuted by* $\mathcal{I}$*?*", where L and R are sets of concept names from the given ontology. We say that an individual *refutes* such an implication question if it is an instance of all concepts in L, and it is an instance of the complement of at least one concept in R. Accordingly, we say that a domain (an ABox) refutes an implication if it contains an individual (instance) that refutes this implication question. Given a DL knowledge base $(\mathcal{T}_0, \mathcal{A}_0)$, and a model $\mathcal{I}$ of this knowledge base that is a formal representation of the application domain, our completion algorithm successively asks questions of the above form. If the expert answers "*no*", then $L \to R$ is not refuted by $\mathcal{I}$ according to the expert's opinion, and thus the GCI $\sqcap L \sqsubseteq \sqcap R$ is added to the current TBox, and the knowledge base is

reclassified. Since $L \rightarrow R$ is not refuted by $\mathcal{I}$, the interpretation $\mathcal{I}$ is still a model of the new TBox obtained this way. If the expert answers "*yes*", then according to the expert's opinion there exists an individual that refutes this question. The expert is asked to extend the current ABox (by adding appropriate assertions on either old or new individual names) such that the extended ABox refutes $L \rightarrow R$ and $\mathcal{I}$ is still a model of this ABox. Upon extension of the ABox, the knowledge base is reclassified and the process continues with the next implication question that does not follow from $\mathcal{T}$ and that is not refuted by $\mathcal{A}$. Once all such questions are answered in this way the knowledge base is complete in a certain well-defined sense [2].

It is possible to optimize this algorithm by employing DL reasoning. Before actually asking the expert whether the implication $L \rightarrow R$ is refuted by $\mathcal{I}$, we can first check whether $\sqcap L \sqsubseteq \sqcap R$ already follows from the current TBox. If this is the case, then we know that $L \rightarrow R$ cannot be refuted by $\mathcal{I}$. Similarly, there are also cases where an implication can be rejected without asking the expert because accepting it would make the knowledge base inconsistent. However, in this case the expert still needs to extend the ABox such that the implication is refuted. The following example illustrates this case, which was not taken into account in the original version of the completion algorithm in [2].

Example 1. Consider the knowledge base $(\mathcal{T}, \mathcal{A})$ with the empty TBox $\mathcal{T} = \emptyset$, and the ABox $\mathcal{A} = \{(\exists r.A \sqcap \forall r.\neg B)(a)\}$. Clearly, the implication $\{A\} \rightarrow \{B\}$ does not follow from $\mathcal{T}$. Moreover, it is not refuted by $\mathcal{A}$ because this ABox does not explicitly contain a named individual that is an instance of both A and $\neg B$. That is, the implication $\{A\} \rightarrow \{B\}$ will be asked to the expert at some step during the execution of the algorithm. If the user accepts this implication, and thus the GCI $A \sqsubseteq B$ is added to $\mathcal{T}$, the knowledge base will become inconsistent since the assertion in $\mathcal{A}$ enforces the existence of an implicit individual that belongs to both A and $\neg B$. This shows that this GCI cannot hold in the intended model $\mathcal{I}$ of $(\mathcal{T}, \mathcal{A})$.

An improved version of the completion algorithm that takes this case into account has been given in [4]. Whenever a new implication question $L \rightarrow R$ that does not already follow from the TBox is generated, before asking the expert this algorithm checks whether addition of the GCI $\sqcap L \sqsubseteq \sqcap R$ will make the TBox inconsistent. If this is the case, it directly asks the expert to provide a counterexample to $L \rightarrow R$. For details of this algorithm the reader is referred to [4]. In the following we demonstrate the execution of this algorithm on a small knowledge base about countries and some of their properties where the intended model $\mathcal{I}$ is the "real world".

Example 2. Assume our knowledge base defines a coastal country as a country that has a border to a sea or an ocean; a Mediterranean country as a country that has border to the Mediterranean Sea; a German-speaking country as a country that has the language German as an official language; and an EU member as a

$\mathcal{A}_{countries}$	Coastal	Mediterranean	EUmember	GermanSpeaking
Italy	+	+	+	−
India	+	−	−	−
Germany	+	−	+	+
Austria	−	−	+	?

Table 1. The ABox before completion

country that is a member of the EU.

$$\begin{aligned}\mathcal{T}_{countries} := \{\ \mathsf{Coastal} &\equiv \mathsf{Country} \sqcap \exists \mathsf{hasBorderTo}.(\mathsf{Ocean} \sqcup \mathsf{Sea}) \\ \mathsf{EUmember} &\equiv \mathsf{Country} \sqcap \exists \mathsf{isMemberOf}.\{\mathsf{EU}\} \\ \mathsf{Mediterranean} &\equiv \mathsf{Country} \sqcap \exists \mathsf{hasBorderTo}.\{\mathsf{MediterraneanSea}\} \\ \mathsf{GermanSpeaking} &\equiv \mathsf{Country} \sqcap \exists \mathsf{hasOfficialLanguage}.\{\mathsf{German}\}\ \}\end{aligned}$$

Table 1 demonstrates the assertions in our ABox $\mathcal{A}_{countries}$ where a + for a concept name C and an individual i means that i is an instance of C, − means that i is an instance of $\neg C$, and a ? means that nothing follows from our knowledge base about membership of i in C. Given this ABox and the TBox, the first implication question posed to the expert is $\{\mathsf{GermanSpeaking}\} \rightarrow \{\mathsf{EUmember}, \mathsf{Coastal}\}$. The answer is "*no*" because in our intended model, which is the real world, Austria is German-speaking, but it is not a coastal country. Assume that the expert turns Austria into a counterexample by asserting that it is a German-speaking country. The second question is then whether $\{\mathsf{GermanSpeaking}\} \rightarrow \{\mathsf{EUmember}\}$ holds. The answer is again "*no*" since Switzerland is a German-speaking country, and it is not an EU member. Assume that the expert adds the new individual Switzerland to the ABox, and asserts that it is an instance of GermanSpeaking and $\neg$EUmember. The next question is $\{\mathsf{Mediterranean}\} \rightarrow \{\mathsf{EUmember}, \mathsf{Coastal}\}$. The answer is again "*no*" because Turkey is a Mediterranean country, but it is not an EU member. Assume that the expert adds the individual Turkey to the ABox, and asserts that it is an instance of $\neg$EUmember. The next question $\{\mathsf{Mediterranean}\} \rightarrow \{\mathsf{Coastal}\}$ follows from the TBox. Thus, it is not posed to the expert, and the algorithm continues with the last question $\{\mathsf{Coastal}, \mathsf{GermanSpeaking}\} \rightarrow \{\mathsf{EUmember}\}$. The answer to this question is "*yes*" because the only countries that are both coastal and German-speaking (Germany and Belgium) are also EU members. Thus the GCI $\mathsf{Coastal} \sqcap \mathsf{GermanSpeaking} \sqsubseteq \mathsf{EUmember}$ is added to the TBox, and the completion process is finished. The completion yields the final context displayed in Table 2.

In [20, 19] Völker and Rudolph have worked on a method that is similar to our method. They have also combined DL reasoning with FCA for ontology refinement, and developed a tool for this purpose. The difference of this work to ours is that, there the main aim is acquiring domain-range restrictions.

	Coastal	Mediterranean	EUmember	GermanSpeaking
Italy	+	+	+	−
India	+	−	−	−
Germany	+	−	+	+
Austria	−	−	+	+
Switzerland	−	−	−	+
Turkey	+	+	−	−

Table 2. The ABox after completion

4 Detecting errors and recovery from errors

Based on the approach described in the previous section, we had implemented a first experimental version of a DL knowledge base completion tool. Our experiments with this tool showed that during completion the expert sometimes makes errors. For better usability of the completion procedure, it is important to support the expert in detecting and correcting these errors.

Detecting Errors

We say that the expert makes an *error* if he extends the knowledge base such that it no longer has the intended model $\mathcal{I}$ as its model. Since the procedure has no direct access to $\mathcal{I}$, in general it cannot detect such errors without help from the expert. The only case where the procedure can automatically detect that an error has occurred is when the knowledge base becomes inconsistent. Obviously, the intended model $\mathcal{I}$ cannot be a model of an inconsistent KB. However, when an inconsistency is detected by DL reasoning, then it is not clear at which stage the actual error was made. In fact, although only the last extension of the knowledge base has made it inconsistent, the deviation from what holds in $\mathcal{I}$ may have occurred in a previous step. Pinpointing [14, 10, 13, 3, 5] can be used to compute all minimal subsets of the knowledge base that are already inconsistent, and thus help the expert to find the place where the error was made. But in the end, the expert needs to tell the completion tool which are the erroneous assertions and/or GCIs.

The expert may also be alerted by DL reasoning to errors in cases where the knowledge base is not inconsistent. In fact, after each extension of the KB, the DL reasoner re-classifies it, i.e., computes the implied subsumption and instance relationships. If one of them contradicts the experts knowledge about $\mathcal{I}$, she also knows that an error has occurred. Again, pinpointing can show all minimal subsets of the knowledge base from which this unintended consequence already follows.

Recovery from Errors

Once the sources of the error are found, the next task is to correct it without producing unnecessary extra work for the expert. Of course, one can just go back

to the step where the first error was made, and continue the completion process from there, this time with a correct answer. The problem with this simple approach is that it throws away all the information about $\mathcal{I}$ that the expert has added (by answering implication queries) after the first error had occurred. Consequently, the completion procedure may again pose implication queries whose answer actually follows from this information. On the DL side, it is no problem to keep those assertions and GCIs that really hold in $\mathcal{I}$. More precisely, the completion procedure can keep the GCIs and assertions for which the expert has stated that they are not erroneous. In fact, our completion procedure allows for arbitrary extensions of the KB as long as the KB stays a model of $\mathcal{I}$.

On the FCA side, it is less clear whether one can keep the implications that have been added after the first error had been made. In fact, since the new (correct) answer differs from the previous incorrect one, the completion procedure may actually produce different implication questions. The proof of correctness of the procedure, as given in [1], strongly depends on the fact that implications are enumerated in some specific lexicographic order. Thus, if we change this order the correctness of the completion algorithm will not be guaranteed any more. In [4] we have overcome this problem by using the previous answers of the expert as "background knowledge" as mentioned in [17], and shown that in this setting the correctness of the algorithm is still guaranteed.

5 OntoComP

Based on our results in [2, 15, 4], we have implemented an open-source ontology completion tool called OntoComP[1], which stands for ONTology COMpletion Plugin. It is written in the Java programming language as a plugin for the Protégé 4 ontology editor [8]. It communicates with the underlying DL reasoner over the OWL API [6]. OntoComP can easily be installed by just copying the `jar` file provided on the project web page into the `plugins` directory of an existing Protégé 4 installation. Upon a new start Protégé will find the OntoComP plugin and open a new tab for it.

In order to complete an ontology with OntoComP, the user first classifies the ontology with a reasoner that is supported by Protégé 4, namely FaCT++ [18] or Pellet [16]. Once the ontology is classified, the OntoComP tab displays the class hierarchy on the left. At this point the user can drag "interesting" class names, which she wants to have in the completion process, from this hierarchy and drop them into the `Context` tab on the right. Simultaneously, the instances of these classes will also be displayed in a table in the `Context` tab. In this table a `+` in row a and column C means that the individual a is an intance of the class C, a `-` means that a is an instance of the complement of C, and a `?` means that nothing is known about the membership of a in class C. When the user is done with selecting the relevant classes, she starts the completion by hitting the *Start* button and sees the first question in the `Messages` tab. If she

[1] available under http://ontocomp.googlecode.com

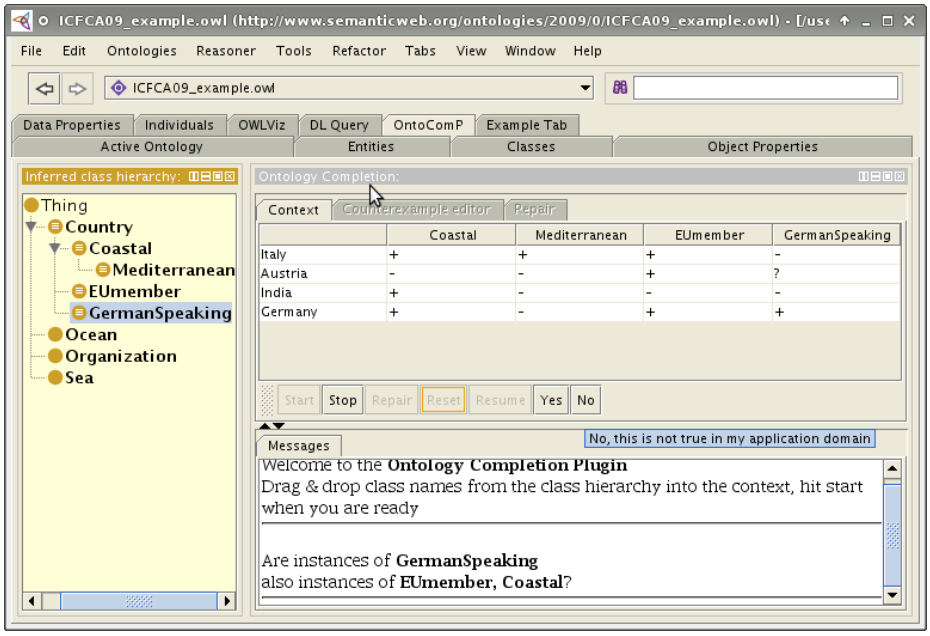

Fig. 2. OntoComP window during completion

confirms the question by hitting the *Yes* button, OntoComP comes up with the next question. If she rejects the question by hitting *No* button OntoComP opens the `Counterexample editor` tab where the user can generate a counterexample to the rejected question. The process continues until all questions have been answered by the user. During completion, at any time the user can suspend the completion process and see her answering history. If she thinks she has made an error at some point, she can repair it by undoing that particular erroneous answer and can continue completion.

5.1 Counterexample Generation.

OntoComP has a counterexample editor for supporting the user during counterexample generation. When the user rejects an implication, OntoComP opens a counterexample editor in a new tab, and displays those individuals from the ABox that can potentially be turned into a counterexample by adding assertions for them. Alternatively, the user can introduce a new individual together with assertions that make it a counterexample. During counterexample generation, OntoComP guides the user and notifies her once she has created a valid counterexample to the implication question.

5.2 Error Recovery.

At any point during knowledge base completion, if the user notices that he has made a mistake in one of the previous steps, he can stop the completion and can request to see the history of all answers he has given. OntoComP displays all the questions asked in previous steps, the answers that were given to these questions, and the counterexamples accompanying negative answers. The user can browse the answering history, correct the wrong answers he has given in the previous steps, and can then continue completion. OntoComP keeps all GCIs and counterexamples that were not marked as incorrect in the knowledge base, i.e., after an error recovery it does not ask the user questions that he has already answered. Pinpointing reasons for consequences (such as inconsistency or unintended subsumption or instance relationships) is not directly integrated into the current version of OntoComP. However, the user could use the pinpointing facilities provided by Protégé 4.

6 Concluding remarks and future work

We have recalled ontology completion, a method which we have introduced previously in [2, 15], and described extensions of it for better usability. We have also described an implementation of it, namely OntoComP, which is a plugin for the Protégé 4 ontology editor. We have described key technologies underlying OntoComP, and discussed how OntoComP makes use of DL reasoning. We have introduced further optimizations that have not taken place originally in [2].

Currently, our method only works for languages that allow for negation. Negation is required while producing counterexamples. Due to increasing popularity of lightweigt DLs like $\mathcal{EL}$ and $\mathcal{EL}^+$, in the future we want to support knowledge bases written in these languages as well. In order to express that an individual is a counterexample to an implication like $L \to R$, we need to say that this individual is an instance of the negation of at least one concept in R. We can overcome this limitation by introducing an additional concept name $\overline{A}$ for every concept name A that takes part in completion, and making A and $\overline{A}$ disjoint. While generating a counterexample, whever the user says that an individual is an intance of $\neg A$, we assert that this individual is an instance of $\overline{A}$.

One other limitation of our method is that after each answer of the expert, the knowlege base has to be reclassified, which can be time consuming for a big real world ontology. In fact, when an expert wants to complete an ontology, she is usually interested in a small fragment of it, and her answers do not necessarily affect the subsumption relations in other fragments of the ontology. Thus, before completion one could in principle extract a module and work on it with our completion method, and after completion "re-plug" this extended module into the original ontology. As future work we are going to investigate this approach and whether it leads to any improvement.

On the practical side we are going to improve our implementation, especially its user interface. Moreover, we are going to test and evaluate OntoComP on large scale ontologies from real world applications.

References

1. F. Baader, B. Ganter, U. Sattler, and B. Sertkaya. Completing description logic knowledge bases using formal concept analysis. LTCS-Report LTCS-06-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2006. See http://lat.inf.tu-dresden.de/research/reports.html.
2. F. Baader, B. Ganter, B. Sertkaya, and U. Sattler. Completing description logic knowledge bases using formal concept analysis. In M. M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 230–235. AAAI Press, 2007.
3. F. Baader, R. Peñaloza, and B. Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}^+$. In J. Hertzberg, M. Beetz, and R. Englert, editors, *Proceedings of the 30th German Conference on Artificial Intelligence (KI2007)*, volume 4667 of *Lecture Notes in Artificial Intelligence*, pages 52–67. Springer-Verlag, 2007.
4. F. Baader and B. Sertkaya. Usability issues in description logic knowledge base completion. In S. Ferré and S. Rudolph, editors, *Proceedings of the 7th International Conference on Formal Concept Analysis, (ICFCA 2009)*, volume 5548 of *Lecture Notes in Artificial Intelligence*, pages 1–21. Springer-Verlag, 2009.
5. F. Baader and B. Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proceedings of the International Conference on Representing and Sharing Knowledge Using SNOMED (KR-MED'08)*, Phoenix, Arizona, 2008.
6. S. Bechhofer, R. Volz, and P. W. Lord. Cooking the semantic web with the OWL API. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *Proceedings of the Second International Semantic Web Conference, (ISWC 2003)*, volume 2870 of *Lecture Notes in Computer Science*, pages 659–675. Springer-Verlag, 2003.
7. B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin, Germany, 1999.
8. M. Horridge, D. Tsarkov, and T. Redmond. Supporting early adoption of OWL 1.1 with Protege-OWL and FaCT++. In *Proceedings of the Second International Workshop OWL: Experiences and Directions (OWLED 2006)*. CEUR-WS, 2006.
9. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: the making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
10. A. Kalyanpur, B. Parsia, E. Sirin, and B. C. Grau. Repairing unsatisfiable concepts in OWL ontologies. In Y. Sure and J. Domingue, editors, *The Semantic Web: Research and Applications. Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, volume 4011 of *Lecture Notes in Computer Science*, pages 170–184. Springer-Verlag, 2006.
11. A. Kalyanpur, B. Parsia, E. Sirin, B. C. Grau, and J. A. Hendler. Swoop: A web ontology editing browser. *Journal of Web Semantics*, 4(2):144–153, 2006.
12. H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The protégé OWL plugin: An open development environment for semantic web applications. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proceedings of the 3rd International Semantic Web Conference, (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 229–243. Springer-Verlag, 2004.
13. T. Meyer, K. Lee, R. Booth, and J. Z. Pan. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In *Proceedings of the 21st Natonal Conference on Artificial Intelligence (AAAI 2006)*, pages 269–274. AAAI Press/The MIT Press, 2006.

14. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 355–362. Morgan Kaufmann, 2003.
15. B. Sertkaya. *Formal Concept Analysis Methods for Description Logics*. Ph.D. dissertation, Institute of Theoretical Computer Science, TU Dresden, Germany, 2007.
16. E. Sirin and B. Parsia. Pellet: An OWL DL reasoner. In *Proceedings of the 2004 International Workshop on Description Logics (DL2004)*, volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.
17. G. Stumme. Attribute exploration with background implications and exceptions. In H.-H. Bock and W. Polasek, editors, *Data Analysis and Information Systems. Statistical and Conceptual approaches. Proceedings of GfKl'95. Studies in Classification, Data Analysis, and Knowledge Organization 7*, pages 457–469. Springer-Verlag, 1996.
18. D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In U. Furbach and N. Shankar, editors, *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer-Verlag, 2006.
19. J. Völker and S. Rudolph. Fostering web intelligence by semi-automatic OWL ontology refinement. In *Proceedings of the 7th International Conference on Web Intelligence (WI)*, 2008.
20. J. Völker and S. Rudolph. Lexico-logical acquisition of OWL DL axioms - An integrated approach to ontology refinement. In R. Medina and S. Obiedkov, editors, *Proceedings of the 6th International Conference on Formal Concept Analysis, (ICFCA'08)*, volume 4933 of *Lecture Notes in Artificial Intelligence*, pages 62–77. Springer-Verlag, 2008.
21. K. Wolstencroft, A. Brass, I. Horrocks, P. W. Lord, U. Sattler, D. Turi, and R. Stevens. A little semantic web goes a long way in biology. In *Proceedings of the 4th International Semantic Web Conference, (ISWC 2005)*, volume 3729 of *Lecture Notes in Computer Science*, pages 786–800. Springer-Verlag, 2005.