Metaproperty-guided Deletion from the Instance-Level of a Knowledge Base

Claudia Schon[1] *, Steffen Staab[1,2], Patricia Kügler[3] *, Philipp Kestel[3], Benjamin Schleich[3] and Sandro Wartzack[3]
{schon,staab}@uni-koblenz.de

[1] Institute for Web Science and Technologies, University of Koblenz-Landau, Germany
[2] Web and Internet Science Research Group, University of Southampton, UK
[3] Engineering Design, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Abstract. The ontology modeling practice of engineering metaproperties of concepts is a well-known technique. Some metaproperties of concepts describe the dynamics of concept instances, i.e. how instances can and cannot be altered. We investigate how deletions in an ontology-based knowledge base interact with the metaproperties *rigidity* and *dependence*. A particularly useful effect are *delete cascades*. We evaluate how rigidity and dependence may guide delete cascades in an engineering application. A case study in the area of product development shows that beyond explicitly defined deletions, our approach achieves *further* automated and desirable deletions of facts with high precision and good recall.

1 Introduction

Ontological metamodeling is a well-known technique which is used in various areas [10,8]. Intuitively, metaproperties allow concepts and roles to be addressed like domain elements and to be assigned to metaclasses or to be linked via meta roles.

In this paper we consider the metaproperties *rigidity* and *dependence* and their interaction with deletions. A property is rigid, if it is essential to all its instances. For example in most scenarios the property of *being a petrol* usually represented as the concept $Petrol$ is rigid, since nothing can stop being a petrol. For the sake of simplicity, we will avoid the term *property* in the following and say that the corresponding *concept* $Petrol$ is rigid. In contrast to this, the concept $SubsidisedFuel$ is usually not rigid, since such a fuel can start or stop being subsidised at any time.

A concept depends on another concept, if for each instance of the first concept there is necessarily an individual of the second concept. For example, concept $PetrolEngine$ depends on concept $Petrol$. In other words: something cannot be a petrol engine without being fueled by some kind of petrol.

Metaproperties not only provide information about concepts or their relationship to one another, but also about dynamic aspects of the instances of a concept and describe if and how they can be changed. For example the rigidity of concept $Petrol$ indicates that in general, it is not desirable to delete the fact that superplus is a petrol from the knowledge base while maintaining all other facts about superplus. Moreover, deleting

* Work supported by DFG EVOWIPE.

the fact that superplus is a petrol removes an essential property of superplus such that one could conclude that superplus no longer exists and should therefore be completely removed from the knowledge base. The dependence metaproperty describes dynamic aspects as well. For example, the fact that a petrol engine depends on the existence of at least one petrol it can be fueled by indicates that in general, deleting these petrols should result in the deletion of the fact that the engine in question is a petrol engine. In accordance with delete cascades in relational databases, we call additional deletions performed because of dependencies *cascading deletions*.

In general, metaproperties in a knowledge base (KB) depend on the conceptualization of the domain. We suggest to account for the variability of conceptualizations by explicitly modeling the context in which the KB is used. We use context-sensitive metaproperties to model this aspect and to take the context into account when determining cascading deletions.

In many domains, cascading deletions are desired by the user. For example, in product development, where the KB representation of a new product model is derived from existing ones, it is necessary to modify the existing model representation to accomodate the requirements of the new product. The reuse of existing models makes it necessary to delete aspects from the KB which are not applicable anymore. The EVOWIPE [4] project is situated in the domain of product development and aims at developing methods to support the product developer in the process of deleting aspects from KBs. In the domain of product development, the KB may harbor many dependencies that can be exploited to maintain the model validity by providing cascading deletions to the product developer freeing him from error-prone manual work [13].

Related work in knowledge representation refers to the deletion of a symbol from a knowledge base as a *forgetting* operation that affects the signature and the formula set of a knowledge base [17], while the notion of *contraction* is used to refer to the consistency-preserving deletion of facts from the knowledge base, which need not necessarily affect the signature [9]. These contraction operators however do not support cascading deletions.

Thus, we assume that when a product developer — or another knowledge representation-lay user — formulates a *delete* operation, intelligent assistance may and should use the ontology and its metamodel in order to guide the deletion process such that a new valid product model is represented in the ensued knowledge base.

In our example, deleting $Petrol(superplus)$ has the following effect: the rigidity of $Petrol$ first leads to the forgetting of individual $superplus$ (by deleting several facts around $superplus$), as well as the cascading deletion of all assertions $PetrolEngine(e)$ for which superplus was the only petrol engine e could be fueled by.

Thus, the main contributions of this paper are:

- The modeling of knowledge about dynamics of concept instances through context-sensitive metaproperties and the use of these metaproperties to obtain operators for additional and cascading deletions. Sect. 3 formalizes several alternative operators to accomplish additional and cascading deletions of instance-level assertions based on metaproperties.

[4] `https://west.uni-koblenz.de/en/research/evowipe`

– An implementation of our approach for OWL [18] KBs with metaproperties stored in annotations and SPARQL update queries to specify the deletions.
– A quantitative evaluation of our approach with a real KB from product development. This evaluation presented in Sect. 4 confirms that the cascading deletions performed by our operators are desired by the engineers working with the KB.

1.1 A Modeling Example from Product Development

Consider a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ that will serve as a running example throughout the paper:

$$\mathcal{T} = \{PetrolEngine \equiv \exists canBeFueledBy.Petrol, \quad (1)$$
$$Manifold \sqsubseteq \exists isConnectedTo.PetrolEngine, \quad (2)$$
$$CatalyticConverter \sqsubseteq \exists isWeldedTo.Manifold, \quad (3)$$
$$ExhaustPipe \sqsubseteq \exists isConnectedTo.Manifold, \quad (4)$$
$$PassengerSeat \sqsubseteq \exists isSecuredBy.SeatBelt\} \quad (5)$$

$$\mathcal{A} = \{PetrolEngine(pte), \quad isConnectedTo(mf, pte),$$
$$Petrol(superplus), \quad ExhaustPipe(ep),$$
$$Petrol(v\text{-}power), \quad isConnectedTo(ep, mf),$$
$$canBeFueledBy(pte, superplus), \quad CatalyticConverter(cc),$$
$$Manifold(mf), \quad isWeldedTo(cc, mf)\}$$

Suppose the KB is used in product development where it is reasonable to assume that the manifold of a car depends on the petrol engine and the exhaust pipe depends on the manifold implying that each individual of concept $Manifold$ is related via role $isConnectedTo$ to an individual of concept $PetrolEngine$. The same applies to individuals of concept $ExhaustPipe$ regarding concept $Manifold$. These dependencies can be modeled by manually adding the *dependence* metaproperty to the concepts $ExhaustPipe$ and $Manifold$. Please note that the dependency cannot be read from the structure of the axioms. For example, although axiom (5) has the same structure as axiom (2), the $PassengerSeat$ is not dependent on $SeatBelt$.

Assume that petrol engine pte is supposed to be replaced by an electric engine. It is reasonable to assume that deleting the petrol engine should result in the deletion of the manifold mf since it depends on the petrol engine which then should result in the deletion of the exhaust pipe ep since it depends on the manifold mf. In accordance with delete cascades in relational databases, we call these additional deletions of $manifold(mf)$ and $exhaustpipe(ep)$ *cascading deletions*. In Sect. 3 we show how to accomplish these cascading deletions with the help of metaproperties.

2 Background and Preliminaries

We introduce syntax and semantics of the description logic (DL) $\mathcal{SHOIN}$ which corresponds to OWL-DL. Given a set of *atomic roles* N_R, the set of *roles* is defined as $N_R \cup \{R^- \mid R \in N_R\}$, where R^- denotes the *inverse role* corresponding to the atomic

role R. A *role inclusion axiom* is an expression of the form $R \sqsubseteq S$, where R and S are roles. A *transitivity axiom* is of the form $Trans(S)$ where S is a role. An RBox $\mathcal{R}$ is a finite set of role inclusion axioms and transitivity axioms. $\sqsubseteq^*$ denotes the reflexive, transitive closure of $\sqsubseteq$ over $\{R \sqsubseteq S, Inv(R) \sqsubseteq Inv(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. A role R is *transitive* in $\mathcal{R}$ if there exists a role S such that $S \sqsubseteq^* R$, $R \sqsubseteq^* S$, and either $Trans(S) \in \mathcal{R}$ or $Trans(Inv(S)) \in \mathcal{R}$. If no transitive role S with $S \sqsubseteq^* R$ exists, R is called *simple*. Let N_C be the set of *atomic concepts* and N_I a set of individuals. The set of *concepts* is inductively defined using the following grammar:

$$C \rightarrow \top \mid \bot \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \geq nS \mid \leq nS \mid \{a\}$$

where $A \in N_C$, C_i are concepts, $R \in N_R$, $S \in N_R$ a simple role and $a \in N_I$.

A *general concept inclusion* (GCI) is of the form $C \sqsubseteq D$ with C and D concepts. A TBox $\mathcal{T}$ is a finite set of GCIs also called axioms. In our setting, an ABox $\mathcal{A}$ is a finite set of assertions of the form $A(a)$ and $R(a, b)$, with A an atomic concept, R an atomic role and a, b are individuals from N_I. A knowledge base (KB) $\mathcal{K}$ is a triple $(\mathcal{R}, \mathcal{T}, \mathcal{A})$ with signature $\Sigma = (N_C, N_R, N_I)$. The tuple $\mathcal{I} = (\cdot^{\mathcal{I}}, \Delta^{\mathcal{I}})$ is an *interpretation* for $\mathcal{K}$ iff $\Delta^{\mathcal{I}} \neq \emptyset$ and $\cdot^{\mathcal{I}}$ assigns an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to each individual a, a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to each atomic concept A, and a relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each atomic role R. $\cdot^{\mathcal{I}}$ then assigns values to more complex concepts and roles as described in Tab. 1. $\mathcal{I}$ is a *model* of $\mathcal{K}$ ($\mathcal{I} \models \mathcal{K}$) if it satisfies all axioms and assertions in $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{A}$ as shown in Tab. 1. If there is no model for $\mathcal{K}$, $\mathcal{K}$ is called inconsistent. An assertion A of the form $B(a)$ or $R(a, b)$ is entailed by a KB $\mathcal{K}$, denoted by $\mathcal{K} \models A$, iff $\mathcal{I} \models A$ for all models $\mathcal{I}$ of $\mathcal{K}$.

2.1 Justification-based Deletion

Since this paper only considers deletions of ABox assertions, we restrict the following definitions to ABox assertions. In the following, $\mathcal{A}_d$ denotes the set of ABox assertions that are supposed to be deleted. When deleting $\mathcal{A}_d$ from a KB, it is not sufficient to only remove all elements contained in $\mathcal{A}_d$ from the ABox since even after their removal they might still be entailed by the KB.

Definition 1 (Justification [11]). *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a KB and α be an ABox assertion. $\mathcal{J} \subseteq \mathcal{A}$ is a* justification *for α in $\mathcal{K}$ if $(\mathcal{R}, \mathcal{T}, \mathcal{J}) \models \alpha$ and for all $\mathcal{J}' \subset \mathcal{J}$, $(\mathcal{R}, \mathcal{T}, \mathcal{J}') \not\models \alpha$. The set of all justifications for α in $\mathcal{K}$ is denoted by $Just(\alpha, \mathcal{K})$.*

To accomplish the deletion of $\mathcal{A}_d$ from a KB $\mathcal{K}$, we suggest to follow [19] and use root justifications.

Definition 2 (Root Justification [19]). *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a KB, $\mathcal{A}_d = \{\alpha_1, \ldots, \alpha_n\}$ a set of ABox assertions and $Just(\alpha_i, \mathcal{K})$ the set of all justifications of α_i in $\mathcal{K}$. A set $J \in \cup_{i=1}^{n} Just(\alpha_i, \mathcal{K})$ is a* root justification *for $\mathcal{A}_d$ in $\mathcal{K}$ iff there is no $J' \in \cup_{i=1}^{n} Just(\alpha_i, \mathcal{K})$ with $J' \subset J$.*

Since the set of all root justifications for $\mathcal{A}_d$ in $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ corresponds to the set of all minimal subsets of $\mathcal{A}$ which together with $\mathcal{R}$ and $\mathcal{T}$ imply an assertion in $\mathcal{A}_d$, it is sufficient to delete exactly one element from each root justification in order to prevent any assertion in $\mathcal{A}_d$ from being entailed. This corresponds to the construction of a minimal hitting set for the set of all root justifications for $\mathcal{A}_d$ in $\mathcal{K}$.

Concepts and Roles	
$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$	$\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$
$\bot^{\mathcal{I}} = \emptyset$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y : (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}$
$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y : (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$(\geq n\ S)^{\mathcal{I}} = \{x \mid \lvert\{y \mid (x,y) \in S^{\mathcal{I}}\}\rvert \geq n\}$
$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$(\leq n\ S)^{\mathcal{I}} = \{x \mid \lvert\{y \mid (x,y) \in S^{\mathcal{I}}\}\rvert \leq n\}$
$(R^{-})^{\mathcal{I}} = \{(y,x) \mid (x,y) \in R^{\mathcal{I}}\}$	
TBox & RBox axioms	**ABox assertion**
$C \sqsubseteq D \Rightarrow C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	$A(a) \Rightarrow a^{\mathcal{I}} \in A^{\mathcal{I}}$
$R \sqsubseteq S \Rightarrow R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$R(a,b) \Rightarrow (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
$Trans(R) \Rightarrow (R^{\mathcal{I}})^{+} \subseteq R^{\mathcal{I}}$	

Table 1: Model-theoretic semantics of $\mathcal{SHOIN}$. R^{+} is the transitive closure of R.

Definition 3 (Hitting Set). *Let $S = \{S_1, \ldots S_n\}$ be a set of sets. A hitting set for S is a set $H \subseteq \cup_{i=1}^{n} S_i$ with $H \cap S_i \neq \emptyset, \forall 1 \leq i \leq n$. If no proper subset of H is a hitting set for S, H is called a minimal hitting set.*

Definition 4 (Deletion). *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a KB and $\mathcal{A}_d$ a set of ABox assertions. Let furthermore J be the set of all root justifications for $\mathcal{A}_d$ in $\mathcal{K}$. A* deletion *Del of $\mathcal{A}_d$ in $\mathcal{K}$ is a minimal hitting set of J.*

Intuitively, a deletion Del of $\mathcal{A}_d$ in $\mathcal{K}$ is a minimal subset of the ABox $\mathcal{A}$ such that no element in $\mathcal{A}_d$ is entailed by $(\mathcal{R}, \mathcal{T}, \mathcal{A} \setminus Del)$.

In general, there can be several minimal hitting sets for the set of all root justifications for the assertions in $\mathcal{A}_d$. Each of these hitting sets corresponds to a deletion of $\mathcal{A}_d$ in $\mathcal{K}$. Choosing one specific deletion can be done by using various semantics [2].

3 Metaproperty-guided Deletion

We now address the task to perform cascading deletions of ABox assertions from a KB. We will use the context-sensitive metaproperties rigidity and dependence to guide deletion and to achieve the desired cascading behavior. These metaproperties have to be added manually to the KB.

3.1 Context-sensitive Metaproperties

A concept is called *rigid*, if it is essential to all its individuals. For instance, in most scenarios the concept $Person$ is rigid, since one cannot stop being a person. In contrast to that, the concept $Student$ is not rigid, because one can start or stop being a student at any time. We adapt this notion of rigidity to DL KBs and specify this metaproperty such that it has a certain scope of validity which we denote by *context*.

Definition 5 (Set of Rigid Concepts, Rigid Assertions). *For a KB $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with signature $\Sigma = (N_C, N_R, N_I)$ and a context ct. The set $Rigid(\mathcal{K}, ct) \subseteq N_C$ denotes the set of rigid concepts in context ct. An ABox assertion stating that an individual belongs to a concept that is element of $Rigid(\mathcal{K}, ct)$ is called* rigid assertion *w.r.t. ct.*

If the KB $\mathcal{K}$ is clear, we slightly abuse notation such that for a set of ABox assertions S, $Rigid(S, ct)$ denotes the set of rigid assertions in S w.r.t. context ct.

In the scope of this paper, we consider concept C to be *dependent* on concept D, if for all instances c of C there necessarily exists an instance d of D. As an example, [10] uses the concept of a *Parent* which is dependent on the concept *Child*. In other words: one cannot be a parent without having a child. We adapt this notion of dependency to description logic KBs and specify these dependencies w.r.t. a certain contex.

Definition 6 (Set of Dependencies / Violated Dependencies). *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a KB and ct a context identifier. In context ct concept C depends on concept D w.r.t. role R in $\mathcal{K}$, if in context ct for every individual c with $\mathcal{K} \models C(c)$ there necessarily exists an individual $d \in Ind(\mathcal{A})$ with $\mathcal{K} \models D(d)$ and $\mathcal{K} \models R(c, d)$. For a context ct, the set of dependencies in $\mathcal{K}$ is given as*

$$Dep(\mathcal{K}, ct) = \{(C, D, R) \mid \textit{in context } ct \textit{ concept } C \textit{ depends on concept } D \textit{ w.r.t. role } R \textit{ in } \mathcal{K}\}.$$

The set of assertions violating a dependency in an KB and a context ct is defined as

$$violatedDep(\mathcal{K}, ct) = \{C(c) \mid (C, D, R) \in Dep(\mathcal{K}, ct) \textit{ and } \mathcal{K} \models C(c) \textit{ and } \neg\exists d \in Ind(\mathcal{A}) \textit{ with } \mathcal{K} \models R(c, d) \textit{ and } \mathcal{K} \models D(d)\}$$

Please note that, one concept may be dependent on another concept in one context and not in another.

3.2 Requirements for Metaproperty-guided Deletion

Rigidity provides information about the epistemic status of concepts, which can be used to understand the dynamics of instances of this concept. Given a knowledge base $\mathcal{K}$ at time t_1 a deletion transaction initiated by a user will cause a transition that makes $\mathcal{K}$ become $\mathcal{K}'$ at time t_2. Metaproperties indicate what may and what should not be an allowed transition. The rules that guide such transitions vary for rigid and non-rigid assertions. Furthermore, these rules take dependencies into account. Systematic analysis of different examples has led to the following requirements for deletion:

1. The deletion of a rigid assertion $C(a)$ should result in *forgetting* individual a from the signature of the KB. This corresponds to removing all assertions containing individual a from the ABox.
2. The deletion of a non-rigid assertion $D(b)$ should result in the *contraction* of $D(b)$ from the KB. This corresponds to determining a deletion for $D(b)$ in the KB and removing this deletion from the KB. The resulting KB can still contain assertions mentioning individual b.
3. The deletion of an assertion should be performed such that the set of rigid assertions removed by the deletion is minimal w.r.t set inclusion.
4. The deletion should have a cascading behavior: a deletion can lead to the violation of dependencies leading to further deletions, which in turn can violate dependencies, and so on. These deletions caused by dependencies are called *cascading deletions*.

Please note that cascading deletions are fundamentally different from deleting inferred assertions. The assertions that are removed by cascading deletion are not assertions that could be inferred, but assertions that must be deleted due to violated dependencies.

3.3 Rigidity-guided Deletion

We now introduce rigidity-guided deletion which considers the rigidity metaproperty and fulfills requirements 1, 2 and 3.

Example 1. Consider the KB given in Sect. 1.1. Suppose this KB is used in the context of product development pd and assume that $Petrol$ is a rigid concept in this context. Hence, $Petrol \in Rigid(\mathcal{K}, pd)$. Assume $PetrolEngine(pte)$ is supposed to be deleted i.e. $\mathcal{A}_d = \{PetrolEngine(pte)\}$. This deletion can be accomplished by deleting one of the two following sets from the ABox, which both constitute a deletion of $\mathcal{A}_d$ from $\mathcal{K}$:

$$Del_1 = \{PetrolEngine(pte), Petrol(superplus)\} \tag{6}$$

$$Del_2 = \{PetrolEngine(pte), canBeFueledBy(pte, superplus)\} \tag{7}$$

The rigidity of $Petrol$ indicates that it is preferable not to delete assertions using this concept. Intuitively, it makes sense not to remove the information that $superplus$ is a petrol but rather the information that pte can be fueled by petrol. Which leads to choosing Del_2 and fulfills requirement 3.

Let us now consider a different scenario, where for some reasons we really want to delete the fact that $superplus$ is a petrol, meaning $\mathcal{A}_d = \{Petrol(superplus)\}$. The rigidity of concept $Petrol$ indicates that an essential property of individual $superplus$ is supposed to be deleted. Therefore, we suggest to entirely remove $superplus$ from KB. The result is the deletion of $Petrol(superplus)$ and $canBeFueledBy(pte, superplus)$. Please note that the deletion of $canBeFueledBy(pte, superplus)$ is not necessary to prevent $Petrol(superplus)$ from being entailed but rather constitutes an additional deletion caused by the rigidity metaproperty of $Petrol$. This fulfills requirement 2.

The rigidity-guided deletion puts these ideas into practice by minimizing the set of removed rigid assertions and forgetting individuals occurring in rigid assertions which will be deleted.

Definition 7 (Rigidity-Guided Deletion). *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a KB, ct be a context and $\mathcal{A}_d$ the set of ABox assertions supposed to be deleted from $\mathcal{K}$. Let Del be a deletion of $\mathcal{A}_d$ with minimal $Rigid(Del, ct)$ w.r.t. set inclusion. A* rigidity-guided deletion *of $\mathcal{A}_d$ from $\mathcal{K}$ in context ct is*

$$\mathcal{K}^{rig}_{\mathcal{A}_d} = \big(\mathcal{R}, \mathcal{T}, (\mathcal{A} \setminus Del) \setminus RigidDel(\mathcal{A}, Del, ct)\big)$$

with

$$RigidDel(\mathcal{A}, Del, ct) = \{D(a), R(a,b), R(b,a) \in \mathcal{A} \mid \exists C(a) \in Rigid(Del, ct)\}.$$

Please note that there can be more than one deletion Del with minimal $Rigid(Del, ct)$. In this case, the result of the rigidity-guided deletion is not specified and we suggest to let the user decide which of the solutions best implements her intentions.

3.4 Dependency-guided Deletion

Now we introduce *dependency-guided deletion* which considers the dependency metaproperty and fulfills requirement 4.

Example 2. Consider the KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ given in Sect. 1.1 that could be used in two different contexts namely product development (pd) and a car repair shop (crs). In the context of product development, the manifold depends on the petrol engine. Removing a specific motor in this context is usually only done if a different kind of motor is supposed to be used. Therefore, it is desired that the deletion of the engine results in the deletion of the manifold which might result in further deletions. In the context of a car repair shop, it is usually desirable to replace only those parts for which replacement is absolutely necessary. Deleting a motor in this context usually means that a defective motor is to be replaced by a new motor of the same type. Therefore, it is desirable in this context not to remove the manifold. The fact that the catalytic converter is welded to the manifold leads to dependencies in both contexts, meaning that the removal of the manifold results in the removal of the catalytic converter in both contexts. This leads to the following set of dependencies:

$$Dep(\mathcal{K}, pd) = \{(PetrolEngine, Petrol, canBeFueledBy), \tag{8}$$
$$(Manifold, PetrolEngine, isConnectedTo), \tag{9}$$
$$(ExhaustPipe, Manifold, isConnectedTo), \tag{10}$$
$$(CatalyticConverter, Manifold, isWeldedTo)\} \tag{11}$$
$$Dep(\mathcal{K}, crs) = \{(CatalyticConverter, Manifold, isWeldedTo)\} \tag{12}$$

We now consider a product developer who wants to replace the petrol engine by an electric engine. For this, he wants to delete $PetrolEngine(pte)$ leading to $\mathcal{A}_d = \{PetrolEngine(pte)\}$. To prevent $\mathcal{A}_d$ from being entailed, one of the two sets Del_1 given in (6) and Del_2 given in (7) presented in Ex. 1 have to be deleted from the ABox. Suppose we choose Del_2. Taking a closer look at the ABox reveals that after deleting $PetrolEngine(pte)$ from the ABox, manifold mf is not connected to an individual of concept $PetrolEngine$ anymore. Due to the open world semantics, the fact that there is no petrol engine connected to the manifold mf explicitly mentioned in the ABox does not contradict $Manifold(mf)$, which is still contained in the ABox. We argue that this might not correspond to the product developer's intention. We suggest so called *dependency-guided* deletion which uses the dependence metaproperty specified in the current context in order to determine additional deletions which are likely to be intended by the user. In our case, we delete $Manifold(mf)$ as well, since the deletion of $PetrolEngine(pte)$ violated dependency (9). Furthermore, dependency-guided semantics has a cascading behavior. In our example, this means that the deletion of $Manifold(mf)$ leads to the violation of dependencies (10) and (11) resulting in the deletion of both $ExhaustPipe(ep)$ and $CatalyticConverter(cc)$.

Overall, the deletion of $PetrolEngine(pte)$ in the context of product development leads to the deletion of the following set of assertions:

$$\{PetrolEngine(pte), canBeFueledBy(pte, superplus), Manifold(mf),$$
$$ExhaustPipe(ep), CatalyticConverter(cc)\}$$

Next consider a mechanic in a car repair shop who wants to replace a broken petrol engine by a new petrol engine of the same type leading to deletion of $PetrolEngine(pte)$ in the context crs. Since the deletion of $\{PetrolEngine(pte)\}$ does not violate any dependencies in this context, no cascading deletions are performed.

Next we define the $Casc$-operator. Given a certain context, a KB $\mathcal{K}$ and a set of assertions $\mathcal{A}_d$ that are supposed to be deleted, the $Casc$-operator computes a set of ABoxes. Each of these ABoxes constitutes a possible result of the cascading deletion for the context under consideration.

Definition 8 ($Casc$-operator). *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a KB, ct a context and $\mathcal{A}_d$ a set of assertions that are supposed to be deleted from $\mathcal{K}$. Then*

$$Casc_1(\mathcal{K}, ct, \mathcal{A}_d) = \{\mathcal{A} \setminus Del \mid Del \text{ a deletion for} \mathcal{A}_d \text{ in } \mathcal{K}\}$$

For $n \in \mathbb{N}$, $n \geq 1$

$$Casc_{n+1}(\mathcal{K}, ct, \mathcal{A}_d) = \{S \setminus Del \mid S \in Casc_n(\mathcal{K}, ct, \mathcal{A}_d) \text{ and } Del \text{ a deletion for } ViolatedDep((\mathcal{R}, \mathcal{T}, S), ct) \text{ in } (\mathcal{R}, \mathcal{T}, S)\}$$

There is always an $i > 0$ such that $Casc_i(\mathcal{K}, ct, \mathcal{A}_d) = Casc_j(\mathcal{K}, ct, \mathcal{A}_d)$ for all $j \geq i$. For convenience, for this i we set $Casc(\mathcal{K}, ct, \mathcal{A}_d) = Casc_i(\mathcal{K}, ct, \mathcal{A}_d)$.

Definition 9 (Dependency-Guided deletion). *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a KB, ct a context, and $\mathcal{A}_d$ a set of assertions that is supposed to be deleted from $\mathcal{K}$. Then $\mathcal{K}^{dep}_{\mathcal{A}_d} = (\mathcal{R}, \mathcal{T}, \mathcal{A}^{dep})$ with $\mathcal{A}^{dep}$ the largest (w.r.t. set inclusion) set in $Casc(\mathcal{K}, ct, \mathcal{A}_d)$ is a* dependence-guided deletion *of $\mathcal{A}_d$ from $\mathcal{K}$ in context ct.*

In general there can be more than one element which is maximal w.r.t. set inclusion in $Casc(\mathcal{K}, ct, \mathcal{A}_d)$. In this case, the dependency-guided deletion is not specified and we suggest to let the user decide which of the solutions best implements her intentions.

The combination of the rigidity and dependency metaproperties will be investigated in the next section.

3.5 Cascading Deletions

The operator for *cascading deletion* considers both the rigidity and the dependence metaproperty and fulfills all four requirements given in Sect. 3.2.

Combining the behavior of rigidity- and dependency-guided deletions does not only add the deletions performed by each individual deletion but interactions between rigid concepts and dependencies can lead to further deletions: the cascading deletions caused by the dependency-guided deletion can lead to the deletion of rigid assertions which leads to the deletion of all assertions containing certain individuals. This can result into the violation of dependencies.

Recall that by Def. 7 for a context ct, $RigidDel(\mathcal{A}, Del, ct)$ denotes the set of all assertions in the $\mathcal{A}$ containing an individual which occurs in a rigid assertion in Del.

Definition 10 (**$Casc^{rd}$-operator).** *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a KB, ct a context, and $\mathcal{A}_d$ a set of assertions that is supposed to be deleted from $\mathcal{K}$. Then*

$$Casc_1^{rd}(\mathcal{K}, ct, \mathcal{A}_d) = \{(\mathcal{A} \setminus Del) \setminus RigidDel(\mathcal{A}, Del, ct) \mid Del \text{ a deletion of } \mathcal{A}_d \text{ in } \mathcal{K}\}$$

For $n \in \mathbb{N}$, $n \geq 1$

$$Casc_{n+1}^{rd}(\mathcal{K}, ct, \mathcal{A}_d) = \{S \setminus Del' \mid S \in Casc_n^{rd}(\mathcal{K}, ct, \mathcal{A}_d) \text{ and } Del \text{ a deletion of } ViolatedDep((\mathcal{R}, \mathcal{T}, S), ct) \text{ in } (\mathcal{R}, \mathcal{T}, S) \text{ and } Del' = Del \cup RigidDel(S, Del, ct)\}$$

There is always an $i > 0$ such that $Casc_i^{rd}(\mathcal{K}, ct, \mathcal{A}_d) = Casc_j^{rd}(\mathcal{K}, ct, \mathcal{A}_d)$ for all $j \geq i$. For convenience, for this i we set $Casc^{rd}(\mathcal{K}, ct, \mathcal{A}_d) = Casc_i^{rd}(\mathcal{K}, ct, \mathcal{A}_d)$.

Definition 11 (Cascading Deletion). *Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a KB, ct a context, and $\mathcal{A}_d$ a set of assertions that is supposed to be deleted from $\mathcal{K}$. Then $\mathcal{K}_{\mathcal{A}_d}^{rd} = (\mathcal{R}, \mathcal{T}, \mathcal{A}^{rd})$ with $\mathcal{A}^{rd}$ the largest (w.r.t. set inclusion) set in $Casc^{rd}(\mathcal{K}, ct, \mathcal{A}_d)$ is a* cascading deletion *of $\mathcal{A}_d$ from $\mathcal{K}$ in context ct.*

In general there can be more than one element that is maximal w.r.t. set inclusion in $Casc^{rd}(\mathcal{K}, ct, \mathcal{A}_d)$. In this case, the cascading deletion is not specified and we suggest let the user decide which of the solutions best implements her intentions.

Example 3. Consider the KB given in Ex. 1 together with the dependencies stated in Ex. 2 and the following set of rigid concepts:

$$Rigid(\mathcal{K}, pd) = \{Petrol, PetrolEngine, Manifold, ExhaustPipe, CatalyticConverter\}$$

Consider a product developer who is confronted with the political decision that, for environmental reasons, $superplus$ is no longer allowed to be used as petrol for newly developed cars. Therefore, he wants to delete $Petrol(superplus)$ leading to $\mathcal{A}_d = \{Petrol(superplus)\}$. In the first step, the rigidity of $Petrol$ leads to the deletion of both $Petrol(superplus)$ and $canBeFueledBy(pte, superplus)$. In the resulting ABox, dependency (8) is violated since there is no petrol left that can be used to fuel petrol engine pte. Therefore, in the next step $PetrolEngine(pte)$ is deleted. The rigidity of concept $PetrolEngine$ causes individual pte to be entirely removed from the ABox leading to the deletion of $PetrolEngine(pte)$ and $isConnectedTo(mf, pte)$. Again dependencies are violated which leads to further cascading deletions. The overall result of the $Casc^{rd}$-operator is $Casc^{rd}(\mathcal{K}, ct, \mathcal{A}_d) = \{\{Petrol(v\text{-}power)\}\}$. The ABox in $Casc^{rd}(\mathcal{K}, ct, \mathcal{A}_d)$ constitutes the ABox resulting from the cascading deletion.

3.6 Design Decisions

In belief revision, a deletion operator should usually fulfill the so-called success postulate, which states that deletion of a non-tautological statement from a KB results in a KB that does not entail the statement. This is only possible, if consistency is established.

This is why we decided to design the metaproperty-guided deletion operators such that they remove all inconsistencies present in a KB even if they have nothing to do with the deletion actually performed. Following this line, we decided to design the operators for both dependency-guided deletion and the cascading deletion such that they delete assertions that have already violated dependencies in the original ABox and even if they have nothing to do with the deletion performed. This behavior of the operators is desired in our project's area of application of product development and is intended to support the product developer. Other areas of application in which this behavior is not desired are conceivable. For this, the deletion operators would have to be adjusted.

3.7 Implementation

We have implemented the three deletion operators as an intelligent assistant[5] which uses SPARQL update queries (with empty insert statement) to specify the deletions. Our implementation uses Pellet [22] for the computation of justifications.

4 Evaluation

To evaluate if the cascading deletions are user-intended, we performed a case study[6]. with a KB $\mathcal{K}$ that formalizes a test rig. Engineers developed the KB [13] in the EVOWIPE project. The KB contains 15 concepts, 17 roles and 26 individuals. After a training on the metaproperties dependence and rigidity provided by us, the engineers decided where to add these metaproperties to their ontology. In the resulting KB, two concepts carry the rigidity metaproperty and 8 dependencies are stored (as annotations of concepts).

To evaluate if the deletions performed by the cascading deletion are desired, a questionnaire consisting of 40 questions of the form presented in Fig. 1 was used. We systematically created the questions for the survey. In general, we observed that:

- If the deletion of $D(b)$ leads to the cascading deletion of $C(a)$, then often exists R such that $\mathcal{K} \models R(a, b)$.
- If the deletion of $R(a, b)$ leads to the cascading deletion of $C(a)$, then often exists D such that $\mathcal{K} \models D(b)$.

Of course other cases are conceivable where the cascading deletion performs several cascading steps, but for the systematic creation of the questions in the questionnaire we have restrict ourselves to the two cases mentioned above.

We determined all pairs $(C(a), D(b))$ and $(C(a), R(a, b))$ with $\mathcal{K} \models C(a)$, $\mathcal{K} \models D(b)$ and $\mathcal{K} \models R(a, b)$. This has led to 183 assertion pairs. For 20 of these pairs, the deletion of the second component leads to a cascading deletion of the first component. For all remaining pairs, the deletion of the second component does not affect the first component.

[5] Implementation available at: `https://github.com/Institute-Web-Science-and-Technologies/SparqlUpdater`

[6] The KB and the questionnaire used in the case study are available at: `https://west.uni-koblenz.de/sites/default/files/research/datasets/evaluation_testrig.zip`

For the questionnaire, we selected 11 pairs where the deletion of the second component leads to the cascading deletion of the first component and randomly selected 29 pairs where the deletion or the second component does not have any effect on the first component. We have intentionally oversampled such that the proportion of cascading deletions in the questionnaire is much higher than in the whole set of pairs to prevent subjects from being inclined to always pick the same negative answer.

Results and Discussion Seven experts from product development, two of whom are co-authors of this paper, answered the questionnaire consisting of 40 questions, leading to 280 asked questions out of which 277 were answered. When they answered the questions, the engineers used the KB and their background knowledge. For inter-rater agreement, we counted both 'rather yes' and 'yes' answers as 'yes' and both 'rather no' and 'no' answers as 'no' and computed Fleiss' Kappa for the results. The Fleiss' Kappa value indicates by how much the raters agreement exceeds an agreement if the questionnaires are completed randomly. For our questionnaire, the Fleiss' Kappa value is 0.561. For the interpretation of Kappa values, [14] suggest $\kappa < 0$ corresponds to poor agreement, $0 \leq \kappa < 0.2$ to slight agreement, $0.2 \leq \kappa < 0.4$ to fair agreement, $0.4 \leq \kappa < 0.6$ to moderate agreement, $0.6 \leq \kappa < 0.8$ to substantial agreement and $0.8 \leq \kappa \leq 1$ to (almost) perfect agreement.

For evaluation of precision and recall, we created a gold standard answer of either 'yes' or 'no' for each of the 40 questions by majority vote of the responses. We compared the result of the majority vote to the result of the operator for cascading deletions leading to a precision of 1 and recall of 0.48.

The precision of 1 indicates, that the engineers agreed with all 11 cascading deletions. There is no case were the cascading deletion deletes an assertion and the engineers want to keep this assertion. The comparatively lower recall of 0.48 suggests that there are deletions desired by the engineers which are not performed by our approach. Analyzing the questions where most engineers wanted to delete the suggested assertion but our approach did not perform this deletion reveals three causes for the discrepancies:

- For some questions, our approach did not perform the deletion desired by the engineers because the corresponding metaproperties were not set in the ontology. Since the metaproperties for the case study were set manually during the creation of the ontology, they were forgotten in some places. This was only revealed after analyzing the results of the questionnaire. The engineers are currently working on extracting some of these metaproperties automatically with the help of *revers engineering* which supports a systematic analysis of product structures and can be used to extract some of the dependencies.
- In one case, a modeling error in the KB was revealed: The concept $subfunction$ was not modeled as a subconcept of the $function$ concept. This modeling error prevented the desired deletion from being implemented by our approach.
- In two cases the analysis revealed that the notion of dependency used by our approach is not sufficient. The engineers need counting dependencies in some cases, i.e. something is only a subassembly if it has at least two parts. Our current notion of dependency cannot map such counting dependencies and therefore cannot perform the desired cascading deletions.

Assume the fact that
measuring_normal_forces belongs to class *function*
is supposed to be deleted. In your opinion, should the fact that
calculation(low_rpm_no.)via_spring_rate+valve_position belongs to class *solution_principle*
be deleted as well?
☐ yes ☐ rather yes ☐ rather no ☐ no

Fig. 1: Example question from the questionnaire answered by seven experts from the FAU.

– In some cases, the analysis revealed that the engineers expected the dependencies to be symmetric. For example concept $solution\_principle$ is modeled to be dependent on concept $function$ w.r.t. role $fulfils$. Role $fulfils$ is symmetric to role $isFulfiledBy$. The engineers expected in this case that the fact that concept $function$ depends on concept $solution\_principle$ w.r.t. role $isFulfiledBy$ follows from the KB, which currently is not the case. We think that this can be regarded as a *mutual dependency* which is not yet supported by our current approach.

The engineers added the metaproperties whose absence had been noticed and fixed the modeling error. On the improved KB, the precision of our approach is 1, recall has risen to 0.7.

We want to focus on the observed counting dependencies as well as mutual dependencies in future work in order to be able to cover more of the desired deletions. Furthermore, we will analyze which of the other notions of dependencies mentioned in [21] could be useful in the context of product development.

5 Related Work

Ontological metamodelling has a long tradition and is used in various areas [8]. We build our approach on two of the metaproperties used in the OntoClean methodology [10]. However in [10], the metaproperties are not used for the task of updating KBs.

The problem of belief revision and updating KBs has received much attention in research [2]. Usually, approaches in this area have the goal to perform the desired changes while maintaining as much of the original KB as possible. In the model-based approach the set of models of the KB resulting from a change operation should be as close as possible to the set of models of the original KB [16,12]. Opposed to the model-based approach, in [15], [5] and [4] the number of axioms and assertions changed by the update are supposed to be minimal. In [15], [5] and [4] instance level deletion, insertion and repair are addressed for DL-Lite KBs. [7] addresses the same tasks for $\mathcal{SHI}$ KBs. In all these approaches, the computed deletions are minimal and a cascading behavior is not considered.

With respect to SPARQL update, [1] addresses the problem of handling inconsistencies introduced by SPARQL updates in $DL\text{-}Lite_{\text{RDFS}_\neg}$ which covers RDFS and concept disjointness axioms. Different semantics of SPARQL ABox updates are defined and skillful query-rewriting is used to perform the updates. These rewritings exploit the fact that in $DL\text{-}Lite_{\text{RDFS}_\neg}$ inconsistencies are caused by at most two ABox assertions and furthermore rely on the fact that the ABox is materialized.

The cascading behavior of the dependency-guided deletion introduced in this paper is related to the K-operator used in the autoepistemic description logic $\mathcal{ALCK}$ [6]. Intuitively, the K-operator can be interpreted as 'known to be'. For example the concept $\mathsf{K}ExhaustPipe$ is interpreted as the set of all individuals for which it is known that they belong to the concept $ExhaustPipe$ meaning that only individuals whose membership to concept $ExhaustPipe$ is explicitly stated in the ABox are interpreted such that they belong to $\mathsf{K}ExhaustPipe$. The K-operator can be used to state dependencies like the fact that the manifold depends on the petrol engine as

$$Manifold \sqsubseteq \exists \mathsf{K}isConnectedTo.\mathsf{K}PetrolEngine$$

intuitively meaning that for every manifold a petrol engine has to be explicitly stated which is known to be connected to the manifold. However it is not possible to state that this dependency is only valid in a certain context. Furthermore, the rigidity metaproperty cannot be stated using the K operator. Up till now only few DL-reasoners support the K-operator and those reasoners only support the K-operator within queries whereas the KB itself is not allowed to use the operator.

In [3] truth maintenance systems are used to track deductive dependencies between statements in an RDF store. In contrast to this, the dependencies used in this paper can change depending on the context and do not necessarily constitute deductive dependencies. In relational databases, certain dependencies can be enforced by adding integrity constraints which are checked during udpates. [20] introduces static integrity constraints to OWL making it possible, for example, to add a constraint stating that "every person must have a social security number" to an OWL KB.

6 Conclusion and Future Work

In this paper, we presented three different deletion operators: The rigidity-guided deletion performs additional deletions by forgetting an individual from the KB's signature. The dependency-guided deletion relies on dependencies present in the KB and shows a cascading behavior by removing assertions violating these dependencies. The third operator, called cascading deletion exploits both rigidity and dependence metaproperties. The interaction of these two metaproperties leads to interesting deletion cascades, which provide intelligent assistants to users, e.g. product developers. In a case study in the area of product development we have shown that the cascading deletions performed by our approach are desired by the product developers. The case study revealed that the product developers would like further deletions, the implementation of which we will tackle in the future. To achieve this, we plan to analyze other types of dependencies in KBs and then exploit them for deletion.

Up till now we only considered deletions. However in practice deletions go hand in hand with insertions. Inserting assertions can easily lead to violated dependencies. We want to use these violated dependencies to make suggestions to the user for further insertions. This cannot be accomplished with a local closed world assumption, since it does not allow to consider transitions between different states of a KB. In contrast to that, dependencies could be checked for each insertion and additional insertions preventing the violation of dependencies could be automatically generated and suggested to the user.

References

1. A. Ahmeti, D. Calvanese, A. Polleres, and V. Savenkov. Handling inconsistencies due to class disjointness in SPARQL updates. In *ESWC*, volume 9678 of *LNCS*. Springer, 2016.
2. C. E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *J. Symb. Log.*, 50(2):510–530, 1985.
3. J. Broekstra and A. Kampman. Inferencing and truth maintenance in RDF schema. In *PSSS*, volume 89 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.
4. D. Calvanese, E. Kharlamov, W. Nutt, and D. Zheleznyakov. Updating aboxes in dl-lite. In *AMW*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.
5. G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On instance-level update and erasure in description logic ontologies. *J. Log. Comput.*, 19(5):745–770, 2009.
6. F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An epistemic operator for description logics. *Artif. Intell.*, 100(1-2):225–274, 1998.
7. U. Furbach and C. Schon. Semantically guided evolution of aboxes. In *TABLEAUX*, volume 8123 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2013.
8. D. Gasevic, D. Djuric, and V. Devedzic. *Model Driven Engineering and Ontology Development (2. ed.)*. Springer, 2009.
9. B. C. Grau, E. Kharlamov, and D. Zheleznyakov. Ontology contraction: Beyond the propositional paradise. In *AMW*, volume 866 of *CEUR Workshop Proceedings*, pages 62–74. CEUR-WS.org, 2012.
10. N. Guarino and C. A. Welty. An overview of ontoclean. In *Handbook on Ontologies*, International Handbooks on Information Systems, pages 201–220. Springer, 2009.
11. M. Horridge. *Justification based explanation in ontologies*. PhD thesis, University of Manchester, UK, 2011.
12. E. Kharlamov, D. Zheleznyakov, and D. Calvanese. Capturing model-based ontology evolution at the instance level: The case of dl-lite. *J. Comput. Syst. Sci.*, 79(6):835–872, 2013.
13. P. Kügler, P. Kestel, C. Schon, M. Marian, B. Schleich, S. Staab, and S. Wartzack. Ontology-based approach for the use of intentional forgetting in product development. In D. Marjanovic, M. Storga, N. Pavkovic, N. Bojcetic, and S. Skec, editors, *DESIGN 2018*, 2018.
14. J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1), 1977.
15. M. Lenzerini and D. F. Savo. On the evolution of the instance level of dl-lite knowledge bases. In *Description Logics*, volume 745 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
16. H. Liu, C. Lutz, M. Milicic, and F. Wolter. Foundations of instance level updates in expressive description logics. *Artificial Intelligence*, 175(18):2170–2197, 2011.
17. C. Lutz and F. Wolter. Foundations for uniform interpolation and forgetting in expressive description logics. In *IJCAI*, pages 989–995. IJCAI/AAAI, 2011.
18. D. McGuinness, E. Kendall, J. Bao, and P. Patel-Schneider. OWL 2 web ontology language quick reference guide (second edition). Technical report, W3C, Dec. 2012. http://www.w3.org/TR/2012/REC-owl2-quick-reference-20121211/.
19. K. Moodley. Debugging and repair of description logic ontologies. Master's thesis, University of KwaZulo-Natal, Durban, South Africa, 2010.
20. B. Motik, I. Horrocks, and U. Sattler. Adding integrity constraints to OWL. In *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
21. P. Simons. *Parts: A Study in Ontology*. Clarendon Press, 1987.
22. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.