Consistent Evolution of OWL Ontologies

Peter Haase[1] and Ljiljana Stojanovic[2]

[1] Institute AIFB, University of Karlsruhe, Germany
[2] FZI at the University of Karlsruhe, Germany
pha@aifb.uni-karlsruhe.de, stojanovic@fzi.de

Abstract. Support for ontology evolution is extremely important in ontology engineering and application of ontologies in dynamic environments. A core aspect in the evolution process is the to guarantee consistency of the ontology when changes occur. In this paper we discuss the consistent evolution of OWL ontologies. We present a model for the semantics of change for OWL ontologies, considering structural, logical, and user-defined consistency. We introduce resolution strategies to ensure that consistency is maintained as the ontology evolves.

1 Introduction

Most of the work conducted so far in the field of ontologies has focused on ontology construction issues, which assumes that domain knowledge encapsulated in an ontology does not change over time. However, in a more open and dynamic environment, the domain knowledge evolves continually [5]. These changes include accounting for the modification in the application domain, incorporating additional functionality according to changes in the users' needs, organizing information in a better way, etc.

Ontology evolution can be defined as the timely adaptation of an ontology to the arisen changes and the consistent management of these changes. It is not a trivial process, due to the variety of sources and consequences of changes, it thus cannot be performed manually by the ontology engineer. Therefore, this process needs to be supported by the ontology management system. An important aspect in the evolution process is to guarantee the consistency of the ontology when changes occur, considering the semantics of the ontology change. A formalization of the semantic of change requires a definition of the ontology model together with its change operations, the consistency conditions and rules to enforce these conditions.

There exists a number of languages for ontologies, both proprietary and standards-based. They differ not only in their syntax, but more importantly in their semantics. The OWL ontology language is a standard for representing ontologies on the Web [8]. However, the semantics of change operations for OWL has not been considered so far. In this paper, we focus on the evolution of OWL ontologies. More precisely, we consider the OWL DL language, including sublanguages such as OWL Lite.

The approach presented in this paper builds partly on our previous work in ontology evolution [18], which we adapt towards handling OWL ontologies. The differences are mostly reflected in the ontology consistency definition. As we will show, it does not suffice to define a fixed set of consistency conditions, due to the characteristics of

A. Gómez-Pérez and J. Euzenat (Eds.): ESWC 2005, LNCS 3532, pp. 182–197, 2005.

the various sublanguages and the varying usage contexts. Instead, we define the consistency of OWL ontologies at three different levels: structural, logical, and user-defined consistency.

We further define methods for detecting and resolving inconsistencies in an OWL ontology after the application of a change. Finally, as for some changes there may be several different consistent states of the ontology, we define resolution strategies allowing the user to control the evolution. We exemplarily present resolution strategies for various consistency conditions.

This paper is organized as follows: The ontology evolution process is described in Section 2. In Section 3 we define the notions of ontology, ontology change operations, and the semantics of change. In Sections 4, 5, and 6 we discuss how to detect and resolve structural inconsistency, logical inconsistency and user-defined inconsistency, respectively. Before we conclude, we present an overview of related work.

2 Evolution Process

Ontology evolution can be defined as the timely adaptation of an ontology and consistent management of changes. The complexity of ontology evolution increases as ontologies grow in size, so a structured ontology evolution process is required. We follow the process described in [18]. The process starts with *capturing changes* either from explicit requirements or from the result of change discovery methods. Next, in the *change representation* phase, changes are represented formally and explicitly. *The semantics of change* phase prevents inconsistencies by computing additional changes that guarantee the transition of the ontology into a consistent state. In the *change propagation* phase all dependent artifacts (ontology instances on the Web, dependent ontologies and application programs using the changed ontology) are updated. During the *change implementation* phase required and induced changes are applied to the ontology in a transactional manner. In the *change validation* phase the user evaluates the results and restarts the cycle if necessary.

In this paper we focus on the semantics of change phase. Its role is to enable the resolution of a given ontology change in a systematic manner by ensuring the consistency of the whole ontology. It is realized through two tasks:

- *Inconsistency Detection*: It is responsible for checking the consistency of an ontology with the respect to the ontology consistency definition. Its goal is to find "parts" in the ontology that do not meet consistency conditions;
- *Change Generation*: It is responsible for ensuring the consistency of the ontology by generating additional changes that resolve detected inconsistencies.

The semantics of change phase of the ontology evolution process is shown in Figure 1. Changes are applied to an ontology in a consistent state (c.f. Change Application in Figure 1), and after all the changes are performed, the ontology must remain consistent (c.f. Change Resolution in Figure 1). This is done by finding inconsistencies in the ontology and completing required changes with additional changes, which guarantee the consistency. Indeed, the updated ontology is not defined directly by applying a re-

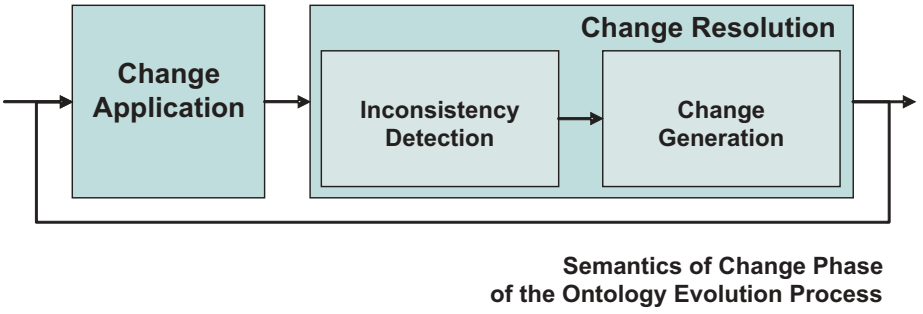

Fig. 1. Semantics of Change Phase

quested change. Instead, it is indirectly characterized as an ontology that satisfies the user's requirement for a change and it is at the same time a consistent ontology.

In this paper we specifically consider the semantics of change phase for OWL DL ontologies. Ontology consistency in general is defined as a set of conditions that must hold for every ontology [18]. Here, we have to distinguish various notions of consistency:

- *Structural Consistency:* First, we have to consider the structural consistency, which ensures that the ontology obeys the constraints of the ontology language with respect to how the constructs of the ontology language are used.
- *Logical Consistency:* Then, we need to consider the formal semantics of the ontology: Viewing the ontology as a logical theory, we consider an ontology as logically consistent if it is satisfiable, meaning that it does not contain contradicting information.
- *User-defined Consistency*: Finally, there may be definitions of consistency that are not captured by the underlying ontology language itself, but rather given by some application or usage context. The conditions are explicitly defined by the user and they must be met in order for the ontology to be considered consistent.

We note that most of the existing evolution systems (including the schema evolution systems as well) consider only the structural consistency. The role of an ontology evolution system is not only to find inconsistencies in an ontology and to alert an ontology engineer about them. Helping ontology engineers notice the inconsistencies only partially addresses the issue. Ideally, an ontology evolution system should be able to support ontology engineers in resolving problems at least by making suggestions how to do that.

Moreover, an inconsistency may be resolved in many ways. In order to help to user to control and customize this process, we have introduced the so-called resolution strategies. Resolution strategies are developed as a method of "finding" a consistent ontology that meets the needs of the ontology engineer. An resolution strategy is the policy for evolution with respect to the his/her requirements. It unambiguously defines the way in which a change will be resolved, i.e. which additional changes will be generated.

In the rest of this paper we formally define different types of consistency and elaborate on how corresponding inconsistencies can be detected and resolved.

3 Ontology Model and Ontology Change Operations

The goal of ontology evolution is to guarantee the correct semantics of ontology changes, i.e. ensuring that they produce an ontology conforming to a set of consistency conditions. The set of ontology change operations – and thus the consistency conditions – depends heavily on the underlying ontology model. Most existing work on ontology evolution builds on frame-like or object models, centered around classes, properties, etc. However, as in this work we focus on the evolution of OWL DL ontologies, we follow the axiom-centered ontology model, heavily influenced by Description Logics. In this section, we will first review the ontology model, define change operations for this model, and describe the semantics of change.

3.1 Ontology Model

OWL DL is a syntactic variant of the $\mathcal{SHOIN}(\mathbf{D})$ description logic [7]. Hence, although several XML and RDF syntaxes exist, for convenience we will adhere to the more compact, traditional $\mathcal{SHOIN}(\mathbf{D})$ syntax. For the correspondence between this notation and various OWL DL syntaxes see [7].

We use a datatype theory $\mathbf{D}$, a set of concept names N_C, sets of abstract and concrete individuals N_{I_a} and N_{I_c}, respectively, and sets of abstract and concrete role names N_{R_a} and N_{R_c}, respectively.

The set of $\mathcal{SHOIN}(\mathbf{D})$ *concepts* is defined by the following syntactic rules, where A is an atomic concept, R is an abstract role, S is an abstract simple role, $T_{(i)}$ are concrete roles, d is a concrete domain predicate, a_i and c_i are abstract and concrete individuals, respectively, and n is a non-negative integer:

$$C \rightarrow A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \geq n\, S \mid \leq n\, S \mid \{a_1, \ldots, a_n\} \mid \\ \mid \geq n\, T \mid \leq n\, T \mid \exists T_1, \ldots, T_n.D \mid \forall T_1, \ldots, T_n.D$$
$$D \rightarrow d \mid \{c_1, \ldots, c_n\}$$

A $\mathcal{SHOIN}(\mathbf{D})$ ontology O is a finite set of axioms of the form[1]: concept inclusion axioms $C \sqsubseteq D$, transitivity axioms $\mathsf{Trans}(R)$, role inclusion axioms $R \sqsubseteq S$ and $T \sqsubseteq U$, concept assertions $C(a)$, role assertions $R(a, b)$, individual (in)equalities $a \approx b$, and $a \not\approx b$, respectively. The common distinction between RBox, TBox and ABox is not relevant for this work. We denote the set of all possible ontologies with $\mathcal{O}$.

Example 1. As a running example, we will consider a simple ontology modelling a small research domain, consisting of the following axioms:
$Researcher \sqsubseteq Person$, $Student \sqsubseteq Person$ (students and researchers are persons), $Article \sqsubseteq Publication$ (articles are publications), $\top \sqsubseteq \forall author^{-}.Publication$, $\top \sqsubseteq \forall author.Person$, (the domain and range of author are publications, persons, resp.), $Article(anArticle)$ (anArticle is an article), $Researcher(peter)$, $Researcher(ljiljana)$ ($peter$ and $ljiljana$ are researchers), $author(anArticle, peter)$, $author(anArticle, ljiljana)$ ($peter$ and $ljiljana$ are authors of $anArticle$).

[1] For the direct model-theoretic semantics of $\mathcal{SHOIN}(\mathbf{D})$ we refer the reader to [9].

3.2 Ontology Change Operation

Based on the ontology model, we can now define ontology change operations.

Definition 1 (Ontology Change Operations). *An* ontology change operation $oco \in OCO$ *is a function* $oco : \mathcal{O} \rightarrow \mathcal{O}$. *Here OCO denotes the set of all change operations.*

For the above defined ontology model of $\mathcal{SHOIN}(\mathbf{D})$, we allow the atomic change operations of adding and removing axioms, which we denote with α^+ and α^-, respectively. Obviously, representing changes at the level of axioms is very fine-grained. However, based on this minimal set of atomic change operations, it is possible to define more complex, higher-level descriptions of ontology changes. Composite ontology change operations can be expressed as a sequence of atomic ontology change operations. The semantics of the sequence is the chaining of the corresponding functions: For some atomic change operations $\text{oco}_1, ..., \text{oco}_n$ we can define $\text{oco}_{composite}(x) = \text{oco}_n \circ ... \circ \text{oco}_1(x) := \text{oco}_n(...(\text{oco}_1))(x)$.

3.3 Semantics of Change

The semantics of change refers to the effect of the ontology change operations and the consistent management of these changes. The consistency of an ontology is defined in terms of consistency conditions, or invariants that must be satisfied by the ontology. We then define rules for maintaining these consistency conditions by generating additional changes.

Definition 2 (Consistency of an Ontology). *We call an ontology* O consistent *with respect to a set of consistency conditions* $\mathcal{K}$ *iff for all* $\kappa \in \mathcal{K}$, O *satisfies the consistency condition* $\kappa(O)$.

At this point, we do not make any restriction with respect to the representation of the consistency conditions. They may be expressed for example as logical formulas or functions. In the following, we will further distinguish between structural, logical and user-defined consistency conditions: $\mathcal{K}_S$, $\mathcal{K}_L$, and $\mathcal{K}_U$, respectively. We will call an ontology *structurally consistent*, *logically consistent* and *user-defined consistent*, if the respective consistency conditions are satisfied for the ontology.

Change Generation. If we have discovered that an ontology is inconsistent, i.e. some consistency condition is not satisfied, we need to resolve these inconsistencies by generating additional changes that lead to a consistent state. These changes are generated by *resolution functions*:

Definition 3 (Resolution Function). *A* resolution function $\varrho \in \mathcal{P}$ *is a function* $\varrho : \mathcal{O} \times OCO \rightarrow OCO$, *which returns for a given ontology and an ontology change operation an additional change operation (which may be composite).*

A trivial resolution function would be a function which for a given ontology and change operation simply returns the inverse operation, which effectively means a rejection of the change. Obviously, for a consistent input ontology, applying a change followed by the inverse change will result in a consistent ontology.

In general, there may be many different ways to resolve a particular inconsistency, i.e. different resolution functions may exist. We can imagine a resolution function that initially generates a set of alternative potential change operations, which may be presented to the user who decides for one of the alternatives. Such a resolution function that depends on some external input is compatible with our definition of a resolution function.

We can now define the notion of a resolution strategy:

Definition 4 (Resolution Strategy). *A* resolution strategy RS *is a total function* $RS : \mathcal{K} \rightarrow \mathcal{P}$ *that maps each consistency condition to a resolution function. Further we require that for all possible ontologies* $O \in \mathcal{O}$ *and for all* $oco \in OCO$ *and all* $\kappa \in \mathcal{K}$*, the assigned resolution function* $\varrho = RS(\kappa)$ *generates changes* $oco' = \varrho(O, oco)$*, which – applied to the ontology* $oco'(O)$ *– result in an ontology that satisfies the consistency condition* κ*.*

The resolution strategy is applied for each ontology change operation in straightforward manner: As long as there are inconsistencies with respect to a consistency condition, we apply the corresponding resolution function.

Please note that a resolution function may generate changes that violate other consistency conditions (resulting in further changes that in turn may violate the previous consistency condition). When defining a resolution strategy, one therefore has to make sure that the application of the resolution strategy terminates, either by prohibiting that a resolution function introduces inconsistencies with respect to any defined consistency condition, or by other means, such as cycle detection.

In the following chapters we will introduce various evolution strategies to maintain the structural, logical and user-defined consistency of an ontology.

4 Structural Consistency

Structural consistency considers constraints that are defined for the ontology model with respect to the constructs that are allowed to form the elements of the ontology (in our case the axioms). However, in the context of OWL ontologies, there exist various sublanguages (sometimes also called species), such as OWL DL, OWL Lite, OWL DLP [19]. These sublanguages differ with respect to the constructs that are allowed and can be defined in terms of constraints on the available constructs. The role of these sublanguages is to be able to define ontologies that are "easier to handle", either on a syntactic level to for example allow easier parsing, or on a semantic level to trade some of the expressivity for decreased reasoning complexity. It is thus important that the ontology evolution process provides support for dealing with defined sublanguages: When an ontology evolves, we need to make sure that an ontology does "not leave its sublanguage".

Because of the variety of the sublanguages, it is not possible to operate with a predefined and fixed set of structural consistency conditions. Instead, we allow to define sublanguages in terms of arbitrary structural consistency conditions along with the corresponding resolution functions that ensure that an ontology change operation does not lead out of the defined sublanguage. Please note that because of the definition of the

ontology model, we do not allow to construct ontologies outside of the OWL DL language.

4.1 Structural Consistency Conditions

We will in the following define what it means for an ontology to be structurally consistent with respect to a certain ontology sublanguage. A sublanguage is defined by a set of constraints on the axioms. Typically, these constraints disallow the use of certain constructs or the way these constructs are used.

Some constraints can be defined on a "per-axiom-basis", i.e. they can be validated for the axioms individually. Other constraints restrict the way that axioms are used in combination. In the following we will show how such consistency conditions can be defined for a particular sublanguage.

Consistency Condition for the OWL Lite sublanguage. OWL Lite is a sublanguage of OWL DL that supports only a subset of the OWL language constructs [2]. We will now show how it can be defined in terms of a set of structural consistency conditions $\mathcal{K}_S$[2]:

- $\kappa_{S,1}$ disallows the use of disjunction $C \sqcup D$,
- $\kappa_{S,2}$ disallows the use of negation $\neg C$,
- $\kappa_{S,3}$ restricts the use of the concept $C \sqcap D$ such that C and D be concept names or restrictions,
- $\kappa_{S,4}$ restricts the use of the restriction constructors $\exists R.C$, $\forall R.C$ such that C must be a concept name,
- $\kappa_{S,5}$ limits the values of stated cardinalities to 0 or 1, i.e. $n \in \{0, 1\}$ for all restrictions $\geq n\, R$, $\leq n\, R$,
- $\kappa_{S,6}$ disallows the usage of the oneOf constructor $\{a_1, \ldots, a_n\}$.

4.2 Resolving Structural Inconsistencies

Once we have discovered inconsistencies with respect to the defined sublanguage, we have to resolve them. An extreme solution would be to simply remove the axioms that violate the constraints of the sublanguage. This would certainly not meet the expected requirements. A more advanced option is to try to express the invalid axiom(s) in a way that is compatible with the defined sublanguage. In some cases, it may be possible to retain the semantics of the original axioms.

Resolution Strategies for OWL Lite. In the following we will present a possible resolution strategy for the OWL Lite sublanguage by defining one resolution function for each of the above consistency conditions in $\mathcal{K}_S$. Although OWL Lite poses many syntactic constraints on the syntax of OWL DL, it is still possible to express complex descriptions using syntactic workarounds, e.g. introducing new concept names and exploiting the implicit negation introduced by disjointness axioms. In fact, using these techniques,

[2] Please note that the constraints for the OWL DL language are already directly incorporated into the ontology model itself.

OWL Lite can fully capture OWL DL descriptions, except for those containing individual names and cardinality restrictions greater than 1 [8].

- $\varrho_{s,1}$ replaces all references to a concept $C \sqcup D$ with references to a new concept name $CorD$, and adds the following axiom: $CorD \equiv \neg(\neg C \sqcap \neg D)$,
- $\varrho_{s,2}$ replaces all references to a concept $\neg C$ in an added axiom with references to a new concept name $NotC$, and adds the following two axioms: $C \equiv \exists R.\top$ and $NotC \equiv \forall R.\bot$, where R is a newly introduced role name,
- $\varrho_{s,3}$ replaces all references to a concept C (or D), where C (or D) is not a concept name or restriction, in concepts $C \sqcap D$ with references to a new concept name aC (or aD), and adds the following axiom: $aC \equiv C$ (or $aD \equiv D$),
- $\varrho_{s,4}$ replaces all references to a concept C (where C is not a concept name) in restrictions $\exists R.C$ or $\forall R.C$ with references to a new concept name aC, and adds the following axiom: $aC \equiv C$.

While these first four resolution functions simply apply syntactic tricks while preserving the semantics, there exist no semantics-preserving resolution functions for the consistency conditions $\kappa_{S,5}$ and $\kappa_{S,6}$.

However, we can either try to approximate the axioms, or in the worst case, simply remove them to ensure structural consistency. We can thus define:

- $\varrho_{s,5}$ replaces all cardinality restrictions $\geq n\, R$ with restrictions $\geq 1\, R$ and removes all axioms containing cardinality restrictions $\leq n\, R$,
- $\varrho_{s,6}$ replaces all occurrences of the concept $\{a_1, \ldots, a_n\}$ with a new concept D and adds assertions $D(a_1), ..., D(a_n)$.

Example 2. Suppose we wanted to add to the ontology from Example 1 the axiom $Publication \sqsubseteq \exists author.\neg Student$, i.e. stating that all publications must have an author who is not a student. As this axiom violates consistency condition $\kappa_{S,2}$, resolution function $\varrho_{s,2}$ would generate a composite change that adds the following semantically equivalent axioms instead: $Publication \sqsubseteq \exists author.NotStudent$, $Student \equiv \exists R.\top$, $NotStudent \equiv \forall R.\bot$, resulting in a structurally consistent ontology.

5 Logical Consistency

While the structural consistency is only concerned about whether the ontology conforms to certain structural constraints, the logical consistency addresses the question whether the ontology is "semantically correct", i.e. does not contain contradicting information.

5.1 Definition of Logical Consistency

The semantics of the $\mathcal{SHOIN}(\mathbf{D})$ description logic is defined via a model-theoretic semantics, which explicates the relationship between the language syntax and the model of a domain: An interpretation $I = (\triangle^I, \cdot^I)$ consists of a domain set $\triangle^I$, disjoint

from the datatype domain $\triangle_{\mathbf{D}}^{I}$, and an interpretation function $\cdot^{I}$, which maps from individuals, concepts and roles to elements of the domain, subsets of the domain and binary relations on the domain, respectively[3]. An interpretation $\mathcal{I}$ satisfies an ontology O, if it satisfies each axiom in O. Axioms thus result in semantic conditions on the interpretations. Consequently, contradicting axioms will allow no possible interpretations.

We can thus define a consistency condition for *logical consistency* κ_L that is satisfied for an ontology O if O is satisfiable, i.e. if O has a model. Please note, that because of the monotonicity of the logic, an ontology can only become inconsistent by adding axioms: If a set of axioms is satisfiable, it will still be satisfiable when any axiom is deleted. Therefore, we only need to check the consistency for ontology change operations that add axioms to the ontology.

Example 3. Suppose, we start out with the ontology from our Example 4.2, i.e. the initial example extended with the axiom $Student \sqsubseteq \neg Researcher$ (Students and Researchers are disjoint). This ontology is logically consistent.

Suppose we now wanted to add the axiom $Student(peter)$, stating that the individual $peter$ is a student. Obviously, this ontology change operation would result in an inconsistent ontology, as we have stated that students and researchers are disjoint on the one hand, and that $peter$ is a student and a researcher on the other hand.

Now, there may be many ways how to resolve this inconsistency. One possibility would be to reject the change $Student(peter)$. Alternatively, we could also remove the assertion $Researcher(peter)$. However, if both of these assertions are correct, the user may not be happy with either decision. The most intuitive one may be to retract the axiom $Student \sqsubseteq \neg Researcher$, but also this may not satisfy the user. A further, more complex change, would be to introduce a new concept $PhdStudent$, which need not be disjoint with researchers.

5.2 Resolving Logical Inconsistencies

In the following, we will present resolution functions that will allow us to define resolution strategies to ensure logical consistency. The goal of these resolution functions is to determine a set of axioms to remove to obtain a logically consistent ontology with "minimal impact" on the existing ontology. Obviously, the definition of minimal impact may be depend on the particular user requirements. A very simple definition could be that the number of axioms to be removed should be minimized. More advanced definitions could include a notion of confidence or relevance of the axioms. Based on this notion of "minimal impact" we can define an algorithm that generates a minimal number of changes that result in a maximal consistent subontology.

However, in many cases it will not be feasible to resolve logical inconsistencies in a fully automated manner. We therefore also present a second, alternative approach for resolving inconsistencies that allows the interaction of the user to determine which changes should be generated. Unlike the first approach, this approach tries to localize the inconsistencies by determining a minimal inconsistent subontology.

[3] For a complete definition of the interpretation, we refer the reader to [7].

Alternative 1: Finding a Consistent Subontology. In our model we assume that the ontology change operations should lead from one consistent ontology to another consistent ontology. If an ontology change operation (adding an axiom, α^+) would lead to an inconsistent ontology, we need to resolve the inconsistency by finding an appropriate subontology $O' \subset O$ (with $\alpha \in O'$) that is consistent. We do this by finding a maximal consistent subontology:

Definition 5 (Maximal consistent subontology). *An ontology O' is a* maximal consistent subontology *of O, if $O' \subseteq O$ and O' is logically consistent and every O'' with $O' \subset O'' \subseteq O$ is logically inconsistent.*

Intuitively, this definition states that no axiom from O can be added to O' without losing consistency. In general, there may be many maximal consistent subontologies O'. It is up to the resolution strategy and the user to determine the appropriate subontology to be chosen.

The main idea is that we start out with the inconsistent ontology $O \cup \{\alpha\}$ and iteratively remove axioms until we obtain a consistent ontology. Here, it is important how we determine which axioms should be removed. This can be realized using a *selection function*. The quality of the selection function is critical for two reasons: First, as we potentially have to search all possible subsets of axioms in O for the maximal consistent ontology, we need to prune the search space by trying to find the *relevant* axioms that cause the inconsistency. Second, we need to make sure that we remove the *dispensible* axioms. (Please note that a more advanced strategy could consider to only remove parts of the axiom.)

The first problem of finding the axioms that cause the inconsistency can be addressed e.g. using a notion of syntactic relevance by analyzing how the axioms are structurally connected:

We can realize a selection function based on *structural connectedness*:

Definition 6 (Connectedness). *Given a set of axioms O, two axioms α and β are* directly structurally connected *– denoted with connected(α, β) –, if there exists an ontology entity $e \in N_C \cup N_{I_a} \cup N_{I_c} \cup N_{R_a} \cup N_{R_c}$ that occurs in both α and β.*

The second problem of only removing dispensable axioms requires more semantic selection functions. These semantic selection functions can for example exploit information about the confidence in the axioms that allows us to remove less probable axioms. Such information is for example available in probabilistic ontology models, such as [4], but will not be considered in this paper.

In the following, we present an algorithm (c.f. Algorithm 1) for finding (at least) one maximal consistent subontology using the definition of structural connectedness (c.f. Definition 6): We maintain a set of possible candidate subontologies Ω, which initially contains only $O \cup \{\alpha\}$ (c.f. line 1), i.e. the consistent ontology O before the change and the added axiom α. In every iteration, we generate a new set of candidate ontologies (line 3) by removing one axiom β_1 from each candidate ontology (line 7) that is structurally connected with α or an already removed axiom (in $O \setminus O'$, line 6), until at least one of the candidate ontologies is a consistent subontology (line 12). The termination is guaranteed based on the fact that once we have removed all axioms from $O \cup \{\alpha\}$ that are transitively connected with α, the ontology again must be consistent

Algorithm 1 Determine consistent subontology for adding axiom α to ontology O

```
1:  Ω := {O ∪ {α}}
2:  repeat
3:    Ω' := ∅
4:    for all O' ∈ Ω do
5:      for all β1 ∈ O' \ {α} do
6:        if there is a β2 ∈ ({α} ∪ (O \ O')) such that connected(β1, β2) then
7:          Ω' := Ω' ∪ {O' \ {β1}}
8:        end if
9:      end for
10:   end for
11:   Ω := Ω'
12: until there exists an O' ∈ Ω such that O' is consistent
```

(provided that α itself is consistent and O was consistent before adding α). As we remove exactly one axiom from each candidate ontology in one iteration, the resulting ontology will not only be maximal with respect to the above definition, but also maximal with respect to cardinality, i.e. the number of axioms in the ontology.

The corresponding resolution function $\varrho_{L,1}$ thus generates changes that remove the minimal set of axioms to ensure consistency: $O \setminus O'$, where O' is the maximal consistent ontology.

Alternative 2: Localizing the Inconsistency. In the second alternative, we do not try to find a consistent subontology, instead we try to find a minimal inconsistent ontology, i.e. a minimal set of contradicting axiom. We call this process *Localizing the inconsistency*. Once we have localized this minimal set, we present it to the user. Typically, this set is considerably smaller than the entire ontology, such that it will be easier for the user to decide how to resolve the inconsistency.

Definition 7 (Minimal inconsistent subontology). *An ontology O' is a* minimal inconsistent subontology *of O, if $O' \subseteq O$ and O' is inconsistent and for all O'' with $O'' \subset O' \subseteq O$, O'' is consistent.*

Intuitively, this definition states that the removal of any axiom from O' will result in a consistent ontology.

Again using the definition of connectedness, we can realize an algorithm (c.f. Algorithm 2) that is guaranteed to find a minimal inconsistent ontology: We maintain a set Ω with candidate ontologies, which initially only consists of the added axiom $\{\alpha\}$ (c.f. line 1). As long as we have not found an inconsistent subontology, we add one structurally connected axiom (line 6) to each candidate ontology (line 7).

Because of the minimality of the obtained inconsistent ontology, it is sufficient to remove any of the axioms to resolve the inconsistency. The minimal inconsistent ontology can be presented to the user, who can select the appropriate axiom to remove.

It may be possible that one added axiom introduced multiple inconsistencies. For this case, the above algorithm has to be applied iteratively.

Algorithm 2 Localize inconsistency introduced by adding axiom α to ontology O

```
1: Ω := {{α}}
2: repeat
3:    Ω' := ∅
4:    for all O' ∈ Ω do
5:       for all β1 ∈ O \ O' do
6:          if there is a β2 ∈ O' such that connected(β1, β2) then
7:             Ω' := Ω' ∪ {O' ∪ {β1}}
8:          end if
9:       end for
10:   end for
11:   Ω := Ω'
12: until there exists an O' ∈ Ω such that O' is inconsistent
```

Example 4. We will now show how Algorithm 2 can be used to localize the inconsistency in our running example, which has been introduced by adding the axiom α $Student(peter)$. Applying the algorithm, we start out with the candidate ontology $\Omega := \{\{Student(peter)\}\}$. Adding the structurally connected axioms, we obtain: $\Omega := \{\{Student(peter), Researcher(peter)\}, \{Student(peter), Student \sqsubseteq Person\}, \{Student(peter), Student \sqsubseteq \neg Researcher\}, \{Student(peter), Student(ljiljana)\}, \{Student(peter), author(anArticle, peter)\}\}$. All of these candidate ontologies are still consistent. In the next iteration, adding the structurally connected axiom $Student \sqsubseteq \neg Researcher$ to the candidate ontology $\{Student(peter), Researcher(peter)\}$ will result in the minimal inconsistent subontology $\{Student(peter), Researcher(peter), Student \sqsubseteq \neg Researcher\}$.

The removal of any of these axioms (which one is to be decided by the user), will lead to a consistent ontology.

6 User-Defined Consistency

The user-defined consistency takes into account particular user requirements that need to be expressed "outside" of the ontology language itself. While an ontology may be structurally consistent (e.g. be a syntactically correct ontology according to a particular OWL sublanguage) and may be logically consistent, it may still violate some user requirements. We can identify two types of user-defined consistency conditions: *generic* and *domain dependent*.

Generic consistency conditions are applicable across domains and represent e.g. best design practice or modeling quality criteria. For example, OntoClean [20] formalizes a set of meta-properties representing the philosophical notions of *rigidity*, *identity*, *unity*, and *dependence*. These meta-properties are assigned to properties (corresponding to concepts in DL terminology) of the ontology. Constraints on the taxonomic relationships define the consistency of the ontology, e.g. a non-rigid property cannot subsume a rigid property.

Domain dependent consistency conditions take into account the semantics of a particluar formalism of the domain. An example are consistency conditions for the OWL-S process model [17] to verify web service descriptions.

In the following we exemplarily show how user-defined consistency conditions and corresponding resolutions function can be described to ensure *modeling quality conditions*. Such modeling quality conditions cover redundancy, misplaced properties, missing properties, etc. We refer the reader to [18] for a complete reference.

One example of redundancy is *concept hierarchy redundancy*. If a direct superconcept of a concept can be reached through a non-direct path, then the direct link is redundant. We can thus define a consistency condition that disallows concept hierrachy redundancy: $\kappa_{U,1}$ is satisfied if for all axioms $C_1 \sqsubseteq C_n$ in O there exist no axioms in O with $C_1 \sqsubseteq C_2$, ..., $C_{n-1} \sqsubseteq C_n$. We can further define a corresponding resolution function $\varrho_{U,1}$ that ensures this consistency condition by generating a change operation that removes the redundant axiom $C_1 \sqsubseteq C_n$.

Example 5. Suppose, we start out with the ontology from our Example 4.2, i.e. the initial example extended with the axiom $Professor \sqsubseteq Person$ (a professor is a person). This ontology is consistent with respect to the consistency definition $\kappa_{U,1}$.

Suppose we now want to add the axiom $Professor \sqsubseteq Researcher$, stating that the a professor is a researcher. Obviously, this ontology change operation would result in an ontology that is inconsistent with respect to $\kappa_{U,1}$ since there is an alternative path (through the concept $Researcher$) between the concept $Professor$ and its direct super-concept $Person$. The resolution function $\varrho_{U,1}$ would generate a change operation that removes the axiom $Professor \sqsubseteq Person$.

7 Related Work

In the last decade there has been very active research in the area of ontology engineering. The majority of research studies in this area are focused on construction issues. However, coping with the changes and providing maintenance facilities require a different approach. There are a very few approaches investigating the problems of inducing changes in ontologies.

[18] defines an ontology evolution process which we have adapted for our work. However, the semantics of change in [18] focuses on the KAON ontology model, which is fundamentally different from the OWL ontology model, as described earlier. A taxonomy of ontology changes for the OWL ontologies can be found in [10]. However, in [10] the the ontology model follows a more object-oriented view, whereas we follow the axiomatic ontology model of [14].

While there exist significant differences between schema evolution and ontology evolution, as elaborated in [13], particular aspects of schema evolution in databases are relevant for our work. [16] provides an excellent survey of the main issues concerned. A sound and complete axiomatic model for dynamic schema evolution in object-based systems is described in [15]. This is the first effort in developing a formal basis for the schema evolution research. The authors define consistency of a schema with a fixed set of invariants or consistency conditions that are tailored to the data model.

However, in the context of OWL ontologies, the notion of consistency is much more mulitfaceted. First, the existing work only considers structural consistency. Not only is the set of structural constraints different due to the difference in the underlying models. We further support the evolution of various fragments (sublanguages) of OWL that are defined using different structural constraints. Furthermore, we consider the notions of logical and user-defined consistency.

Regarding the notion of logical conistency, the research done in belief revision is of interest: Here, the revision problem is concerned about resolving contradictions by minimal mutilation of a set of beliefs. The combination of classical approaches with description logics is subject of ongoing research [6].

Finally, there are several tools that support species validation (corresponding to our structural consistency) or localizing inconsistencies in ontologies. For example, the OWL Protege Plugin [11] provides species validation including explanations where certain problems occurred. The OWL Protege Plugin also provides explanations on ontology changes, i.e. new subsumptions that have been inferred, logical inconsistencies that have been introduced (based on RACER reasoning services). [1] presents a "symptom" ontology describing inconsistencies and errors in ontologies. It provides various levels of severity provides a classification of inconsistencies. However, there is no support for preserving consistency in the case that consistency conditions are violated in the presence of ontology changes.

8 Conclusion and Outlook for Future Work

In this paper we have presented an approach to formalize the semantics of change for the OWL ontology language (for OWL DL and sublanguages in particular), embedded in a generic process for ontology evolution. Our formalization of the semantics of change allows to define arbitrary consistency condititions – grouped in structural, logical, and user-defined consistency – and to define resolution strategies that assign resolution functions to that ensure these consistency conditions are satisfied as the ontology evolves. We have shown examplarily, how such resolution strategies can be realized for various purposes.

The methods described in the previous sections have been implemented on top of KAON2[4], an ontology management system and inference engine that supports a large subset of OWL DL. The implementation includes evolution strategies for various fragments of ontology languages, including OWL DL, OWL Lite, as well evolution strategies for logical consistency. Additionally it allows to plug-in further evolution strategies for structural consistency (to support additional sublanguages), logical consistency, and user-defined consistency. The approach will be evaluated in the context of various ongoing research projects, including SEKT and OntoGov.

In the future we plan to describe on ontology definition meta-model (ODM) in the style of [3] on top of our axiomatic ontology model and to define the semantics of change for the ODM to support users and applications that prefer a more object-oriented-like ontology model.

[4] `http://kaon2.semanticweb.org/`

Acknowledgments

Research reported in this paper has been partially financed by the EU in the IST project SEKT (IST-2003-506826) (http://www.sekt-project.com/) and IST project OntoGov (IST-2002-507237) (http://www.ontogov.com/). We would like to thank our colleagues for fruitful discussions.

References

1. Kenneth Baclawski, Christopher J. Matheus, Mieczyslaw M. Kokar, Jerzy Letkowski, and Paul A. Kogut. Towards a symptom ontology for semantic web applications. In McIlraith et al. [12], pages 650–667.
2. Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language reference. http://www.w3.org/TR/owl-ref/.
3. Sara Brockmans, Raphael Volz, Andreas Eberhart, and Peter Löffler. Visual modeling of owl dl ontologies using uml. In McIlraith et al. [12], pages 198–213.
4. Zhongli Ding and Yun Peng. A Probabilistic Extension to Ontology Language OWL. In *Proceedings of the 37th Hawaii International Conference On System Sciences (HICSS-37).*, Big Island, Hawaii, January 2004.
5. D. Fensel. Ontologies: dynamics networks of meaning. In *Proceedings of the 1st Semantic web working symposium*, Stanford, CA, USA, 2001.
6. Giorgos Flouris. Belief change in arbitrary logics. In *HDMS*, 2004.
7. I. Horrocks and P. F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *Journal of Web Semantics*, 1(4), 2004.
8. 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), 2003.
9. I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.
10. Michel Klein. *Change Management for Distributed Ontologies*. PhD thesis, Free University of Amsterdam, 2004.
11. Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The protégé owl plugin: An open development environment for semantic web applications. In McIlraith et al. [12], pages 229–243.
12. Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors. *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, volume 3298 of *Lecture Notes in Computer Science*. Springer, 2004.
13. N. F. Noy and M. Klein. Ontology evolution: not the same as schema evolution. In *SMI technical report SMI-2002-0926*, 2002.
14. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. Owl web ontology language semantics and abstract syntax. http://www.w3.org/TR/2004/REC-owl-semantics-20040210/.
15. Randel J. Peters and M. Tamer Özsu. An axiomatic model of dynamic schema evolution in objectbase systems. *ACM Trans. Database Syst.*, 22(1):75–114, 1997.
16. John F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393, 1995.

17. L. Stojanovic, A. Abecker, N. Stojanovic, and R. Studer. On managing changes in the ontology-based e-government. In *Proceedings of the 3rd International Conference on Ontologies, Databases and Application of Semantics (ODBASE 2004)*, number 3291 in Lecture Notes in Computer Science, pages 1080–1097, Agia Napa, Cyprus, November 2004. Springer Verlag.
18. Ljiljana Stojanovic. *Methods and Tools for Ontology Evolution*. PhD thesis, University of Karlsruhe, 2004.
19. Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, University of Karlsruhe, 2004.
20. Christopher A. Welty and Nicola Guarino. Supporting ontological analysis of taxonomic relationships. *Data Knowledge Engineering*, 39(1):51–74, 2001.