A Framework for Handling Inconsistency in Changing Ontologies

Peter Haase[1], Frank van Harmelen[2], Zhisheng Huang[2], Heiner Stuckenschmidt[2], and York Sure[1]

[1] Institute AIFB, University of Karlsruhe, Germany
{haase, sure}@aifb.uni-karlsruhe.de
[2] Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
{frankh, huang, heiner}@cs.vu.nl

Abstract. One of the major problems of large scale, distributed and evolving ontologies is the potential introduction of inconsistencies. In this paper we survey four different approaches to handling inconsistency in DL-based ontologies: consistent ontology evolution, repairing inconsistencies, reasoning in the presence of inconsistencies and multi-version reasoning. We present a common formal basis for all of them, and use this common basis to compare these approaches. We discuss the different requirements for each of these methods, the conditions under which each of them is applicable, the knowledge requirements of the various methods, and the different usage scenarios to which they would apply.

1 Introduction

Ontologies in real-world applications are typically not static entities, they evolve over time. One of the major problems of evolving ontologies is the potential introduction of inconsistencies as a result of applying changes. Previous related work includes the definition of evolution strategies to handle inconsistencies for evolving ontologies in a centralized setting (*cf.* [14]) and for the handling of ontology changes in a distributed setting (*cf.* [9]). However, such approaches rely on different assumptions, including different ontology models (in particular they do not consider DL-based ontologies), use different notions for ontology change and inconsistency and typically cover a specific use case.

When dealing with changing ontologies we found four major use cases which require methods for dealing with inconsistencies. First, changing an initially consistent ontology potentially introduces inconsistencies. This typically occurs in settings where one is in control of changes and needs support for maintaining consistency during evolution. Second, re-using ontologies in open settings such as the Web might include the retrieval of inconsistent ontologies that should be fixed before usage. While these use cases typically occur during the development of ontologies, handling of inconsistencies is also relevant during runtime of ontology-based applications as illustrated in the following. Third, in some cases consistency cannot be guaranteed at all and inconsistencies cannot be repaired, still one wants to derive meaningful answers when reasoning. Often this is the case when schema-level and instance-level of ontologies are evolved separately without synchronizing the changes continuously. Fourth, when applying an

Y. Gil et al. (Eds.): ISWC 2005, LNCS 3729, pp. 353–367, 2005.

ontology one faces the challenge to decide whether the usage of other, e.g. newer, versions of this ontology might lead to inconsistencies in an application, or, in other words, whether the versions are compatible with respect to certain aspects. While the former use cases typically occur during the development of ontologies, the latter ones illustrate the handling of inconsistencies during the runtime of ontology-based applications.

In this paper we define a framework for combining currently separate methods for inconsistency-handling in changing ontologies. This framework is based on formally defined notions including *ontology change* and *inconsistency* for DL-based ontologies, thus being in line with the state-of-the-art representation formalism for ontologies OWL [10]. To meet the requirements of the above mentioned use cases our framework consists of the following main components: *consistent ontology evolution* guarantees the continuous consistency of ontologies in the presence of changes by applying evolution strategies; *repairing inconsistencies* fixes ontologies that are already inconsistent; *reasoning with inconsistent ontologies* returns meaningful query results for queries to inconsistent ontologies; finally, *multi-version reasoning* considers not only the latest version of an ontology, but all previous versions as well to deal with inconsistencies that arise from the interaction of the ontology with its environment in terms of instance data and applications.

Core decisions which had to be taken during the definition of our framework include syntactic vs. semantic definitions for ontology changes, functional vs. non-functional notion of change, language dependent vs. language independent definitions and whether one only considers logical properties or also other forms like structural, data, etc.

The main benefit of our framework consists of the identification of typical kinds of problems one actually has when having to deal with inconsistent ontologies and the provision of methods and implementations to deal with the problems. The framework has been implemented (to large extents) as part of the EU project SEKT[1].

The paper is structured as follows. In the next Section 2 we present a general overview of the framework and describe the core decisions which had to be taken during the design of our framework. In Section 3 we describe basic definitions underlying the framework such as the notion of ontology change. The following Section 4 then describes on top of these definitions each of the components for handling of inconsistencies in detail. We compare the different approaches to help identifying which component can be applied in which situation. Before concluding we present related work.

2 General Overview

The study of ontology change management covers a very broad spectrum [11,9,14]. It encompasses methods and techniques necessary to support modifications to ontologies. One important aspect that must be dealt with in a comprehensive treatment of ontology change is handling of inconsistencies. While we may distinguish various forms of inconsistencies (c.f. [2], in this work, we consider ontologies as logical theories. We therefore focus on *logical inconsistencies* in ontologies. We discuss four different approaches to ontology change, and the different implications each of these has for the management of inconsistencies arising from the changing ontologies:

[1] `http://www.sekt-project.com/`

Consistent Ontology Evolution is the process of managing ontology changes by preserving the consistency of the ontology with respect to a given notion of consistency. The consistency of an ontology is defined in terms of consistency conditions, or invariants that must be satisfied by the ontology.

Repairing Inconsistencies involves a process of diagnosis and repair: first the cause (or: a set of potential causes) of the inconsistency needs to be determined, which can subsequently be repaired.

Reasoning with Inconsistent Ontologies does not try to avoid or repair the inconsistency (as in the previous two approaches), but simply tries to "live with it" by trying to return meaningful answers to queries, even though the ontology is inconsistent.

Ontology Versioning manages the relations between different versions of an ontology, and a notion of compatibility with such versions. One such compatability relation is inconsistency: even though two versions of an ontology may each be consistent in themselves, they might derive some opposite conclusions, and would then be mutually inconsistent.

In order to find a common ground for these different approaches to dealing with inconsistencies in changing ontologies, a number of choices have to be made concerning this common ground. We outline the most important of these choices here.

Syntactic or semantic. An obvious essential question is what we count as a change in an ontology? Do we count every syntactic modification to an ontology, or only those syntactic modifications that affect the semantics of the ontology. A simple example to illustrate the difference is to consider the ontology (using DL syntax, c.f. Section 3):

$$C1 \sqsubseteq C2, C1(x), C2(x)$$

Removing the third statement is clearly a syntactic change, but not a semantic one (the set of models of the ontology does not change, since the removed statement is also implied by the remaining two). This choice boils down to that of defining an ontology as a set of axioms (a syntactic object), or as a set of models (a semantic object, typically captured by finite set of axioms). In this paper, we have chosen to define an ontology as a set of axioms, allowing us to capture any syntactic modification to an ontology. We consider the syntactic approach most suitable as the same logical theory can be encoded by different sets of axioms that have different computational properties that are also important in applications (e.g. many ontologies that are formally in OWL-Full can be rephrased into an equivalent ontology in OWL-DL). The syntactic approach enables us to distinguish between these two encodings. This choice is in line with other studies of changing ontologies, e.g. [9,11,14].

Language dependent vs. language independent. A second important choice is the restriction of our definitions to a specific ontology language. It is now commonly accepted that any ontology language should have its foundation on logic. While the approaches we present are in general applicable to any ontology language based on a (monotonic) logic, we pay special attention to the OWL ontology language. As the OWL ontology language has been standardized by the W3C consortium, we will adhere to the underlying OWL ontology model. In particular, we consider the language OWL-DL (which includes sublanguages such as OWL-Lite). OWL-DL is a syntactic variant of the $\mathcal{SHOIN}(\mathbf{D})$ description logic [5]. In the following we will therefore use the more compact, traditional description logic syntax.

Functional or non-functional change. A final important decision is whether we regard ontology change as a deterministic or non-deterministic operation: does any operation on an ontology result in a single well-defined result, or in a set of possible outcomes. Our earlier choice for a syntactic view of ontology change makes it plausible to limit change to a deterministic, functional operation.

3 Basic Definitions

This section describes the basic definitions which involve ontology change and inconsistency processing. Some of these basic definitions may be so obvious or well-known that they may be considered to be trivial. Those terminologies are usually found under different contexts and theories with different meanings and implications. In this paper, we would like to provide a unique framework to define those definitions formally, which can serve as a solid foundation for the theory of ontology change to avoid unnecessary ambiguities on the definitions and minimize the disagreement among the researchers.

In general, an ontology language can be considered to be a set that is generated by a set of syntactic rules. Namely, an ontology can be viewed as a formula set, alternatively called axioms, which involves a set of vocabulary.

Definition 1 (Ontology). *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:*

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

A $\mathcal{SHOIN}(\mathbf{D})$ *ontology* O *is a finite set of axioms of the form*[2]*: 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 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 $\triangle^I_{\mathbf{D}}$, 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. This leads us to the definition of a consistent ontology:

[2] For the direct model-theoretic semantics of $\mathcal{SHOIN}(\mathbf{D})$ we refer the reader to [6].

[3] For a complete definition of the interpretation, we refer the reader to [5].

Definition 2 (Consistent Ontology). *An ontology O is consistent iff O is satisfiable, i.e. if O has a model.*

To be able to define queries against ontologies, we rely on the notion of entailment:

Definition 3 (Entailment). *Given a logical language $\mathcal{L}$, an entailment $\models$ states a relation between an ontology O and an axiom $\alpha \in \mathcal{L}$. Namely, an entailment is a set of pairs $\langle O, \alpha \rangle$. We use $O \models \alpha$ to denote that the ontology O entails the axiom α. Alternatively, we say that α is a consequence of the ontology O under the entailment relation $\models$. The entailment relation is said to be a standard one iff α always holds in any model in which the ontology O holds, i.e., for any model M, $M \models O \Rightarrow M \models \alpha$.*

Usually we use $\models$ and $\not\approx$ to denote a standard entailment and a non-standard entailment respectively if it does not cause any ambiguity. A standard entailment is explosive, namely, any formula is a consequence of an inconsistent ontology. Namely, if an ontology O is not consistent, then for any axiom α, $O \models \alpha$.

A general goal of the approaches proposed in this paper is to obtain consistent query answers. Thus, we have the following definitions.

Definition 4 (Query). *A query with respect to an entailment relation $\models$ is a pair of an ontology O and an axiom α, written '$O \models \alpha?$'.*

Definition 5 (Query Answer). *An answer to a query '$O \models \alpha?$' is a value in the set $\{true, false\}$ as $O \models \alpha$ and $O \not\models \alpha$ respectively.*

When we talk about inconsistency, we usually assume that the existence of a negation operator $\neg$ which can be used to denote the negation of an axiom[4].

Definition 6 (Consistent Query Answer). *For an ontology O and an entailment relation $\not\approx$, an answer '$O \not\approx \alpha$' is said to be consistent if $O \not\approx \neg\alpha$.*

Proposition 1 (Consistent Ontology and Consistent Query Answer). *For a consistent ontology O, its query answer is always consistent under a standard entailment. Namely, the consequence set $\{\alpha : O \models \alpha\}$ is consistent.*

To be able to deal with inconsistent ontologies, the following two definitions are useful:

Definition 7 (Maximal consistent subontology). *An ontology O' is a maximal consistent subontology of O, if $O' \subseteq O$ and O' is consistent and every O'' with $O' \subset O'' \subseteq O$ is 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'.

Definition 8 (Minimal inconsistent subontology). *An ontology O' is a minimal inconsistent subontology of O, if $O' \subseteq O$ and O' is inconsistent and every O'' with $O'' \subset O'$ is consistent.*

[4] In the considered description logic, there exists no universal negation operator. However, negation can be simulated, e.g. to express the negation of the role assertion $\neg R(a, b)$ we can write $\neg(\exists R.\{b\})(a)$.

Finally, we formalize changes to ontologies. As we have argued in the previous section, in the paper we will focus on functional ontology changes. Thus, we have:

Definition 9 (Ontology Change Operation). *An* ontology change operation *oco is a function oco* : $\mathcal{O} \rightarrow \mathcal{O}$.

There might exist many different ontology change operations. In this paper, we will not discuss a list of possible ontology changes. Instead, we consider the two atomic change operations of adding and removing axioms. Other change operations can be defined in terms of those two atomic change operations with different sequences of the executions. 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 $\text{oco}_{composite}(x) = \text{oco}_n \circ ... \circ \text{oco}_1(x) := \text{oco}_n(...(\text{oco}_1))(x)$.

As we have argued in the previous sections, in this paper, we consider only functional and syntactic-based change operations. Accordingly we define the semantics of the change operations: $O\dot{+}\alpha := O \cup \{\alpha\}$ and $O\dot{-}\alpha := O \setminus \{\alpha\}$.

4 Handling Inconsistencies of Changing Ontologies

In Section 2 we have already presented a general overview of the different strategies for handling the problem of inconsistencies in changing ontologies. In the following, we describe these different strategies in terms of the notions introduced in the previous section and provide a comparison.

4.1 Consistent Ontology Evolution

The goal of consistent ontology evolution is to maintain the consistency of ontology in the presence of changes. There are strong forms of guaranteeing consistency that strictly forbid change operations that can lead to an inconsistent ontology. A radical approach is to forbid the use of logical operators that potentially introduce inconsistency (i.e. negation, but also other constructs). The drawback is a substantial loss of expressive power. The strategy that we consider here is to define a semantics of change that ensures consistency by (1) detecting potential inconsistencies caused by changes and (2) generating additional changes for a transition into another consistent state [2]. We can summarize the approach of consistent ontology evolution as follows: For a consistent ontology O and a change operation *oco*, the task of consistent ontology evolution is to generate a change operation oco' such that $O' = oco'(oco(O))$ results in a consistent ontology O'.

Please note that because of the monotonicity of the considered logic, an ontology can only become logically 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.

Effectively, if $O \cup \{\alpha\}$ is inconsistent, in order to keep the resulting ontology consistent some of the axioms in the ontology O have to be removed. In this sense, the add-operation and the remove-operation are similar to the belief revision operation and the belief contraction operation in the theories of belief revision [1].

In the following, we will present strategies to ensure logical consistency. The goal of these strategies is to determine a set of axioms to remove to obtain a logically consistent ontology with "minimal impact" on the existing ontology, e.g. based on Definition 7 of a

maximal consistent subontology. 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 have potentially have to search all possible subsets of axioms in O for a 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.

The first problem of finding the axioms that cause the inconsistency can be targeted by considering that there must be some "connection" between these problematic axioms. We formalize this notion with the following definition.

Definition 10 (Connectedness). *A connection relation $\mathcal{C}$ is a set of axiom pairs, namely, $\mathcal{C} \subseteq \mathcal{L} \times \mathcal{L}$.*

A very simple, but useful connection is that of the direct structural connection relation:

Definition 11 (Direct Structural Connection). *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.*

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 11): We maintain a set of possible candidate subontologies Ω, which initially contains only $O \cup \{\alpha\}$, 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 by removing one axiom β_1 from each candidate ontology that is structurally connected with α or an already removed axiom (in $O \setminus O'$), until at least one of the candidate ontologies is a consistent subontology.

The properties of the algorithm (efficiency, completeness) will depend on the properties of the connectedness relation. The above definition of structural connectedness provides good heuristics to efficiently find a maximal consistent subontology, but is not complete for the case where axioms causing an inconsistency are not structurally connected at all.

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

```
Ω := {O ∪ {α}}
repeat
  Ω' := ∅
  for all O' ∈ Ω do
    for all β1 ∈ O' \ {α} do
      if there is a β2 ∈ ({α} ∪ (O \ O')) such that connected(β1, β2) then
        Ω' := Ω' ∪ {O' \ {β1}}
      end if
    end for
  end for
  Ω := Ω'
until there exists an O' ∈ Ω such that O' is consistent
```

Example 1. We will now show how Algorithm 1 can be used to maintain consistency. As a running example, we will consider a simple ontology modelling a small research domain, consisting of the following axioms:
$O_1 = \{Employee \sqsubseteq Person,\ Student \sqsubseteq Person,\ PhDStudent \sqsubseteq Student,\ Employee \sqsubseteq \neg Student$,[5] $PhDStudent(peter)\}$.

Now consider a change operation oco_1 that adds the axiom $\alpha = PhDStudent \sqsubseteq Employee$. $oco_1(O_1)$ results in an inconsistent ontology.

Algorithm 1 starts with $O_1 \dot{+} \alpha$ as element of the set of potential ontologies. In the first iteration, a set of new potential ontologies is created by removing one of the axioms that are structurally connected with the α. These axioms are: $PhDStudent(peter)$, $Employee \sqsubseteq \neg Student$, $PhDStudent \sqsubseteq Student$ and $Employee \sqsubseteq Person$.

The removal of either $PhDStudent(peter)$, $PhDStudent \sqsubseteq Student$ or $Employee \sqsubseteq \neg Student$ will result in a maximal consistent subontology. For the decision which axiom should be removed from the ontology, one can rely on further background information indicating the relevance of the axioms, or on interaction with the user. For the following examples, we assume that the resulting ontology O_2 is created by removing the axiom $Student \sqsubseteq \neg Employee$, i.e. $O_2 = O_1 \dot{+} PhDStudent \sqsubseteq Employee \dot{-} Student \sqsubseteq \neg Employee$.

4.2 Repairing Inconsistencies

The most straightforward approach to inconsistencies is to repair them when they are detected [13]. Repairing an inconsistency actually consists of two tasks: Locating Inconsistencies and Resolving Inconsistency. The task of repairing inconsistencies can thus be defined as: For an inconsistent ontology O we generate a change operation oco such that $O' = oco(O)$ results in a consistent ontology O'.

Locating Inconsistencies As a first step, the source of the inconsistency has to be detected. Normally, the source is a set axioms that when being part of the model at the same time make it inconsistent.

An algorithm to find a subontology which leads to an unsatisfiable concept (adopted from [13]) can use similar ideas like those for consistent ontology evolution. The main difference is that the latter assumes that the intended minimal inconsistent ontologies would contain an added axiom α, whereas the former has no such requirement but starting with an unsatisfiable concept C for the connection checking[6]. Algorithm 2 uses the increment-reduction strategy to find a minimal subontology for an unsatisfiable concept. Namely, the algorithm finds a subset of the ontology in which the concept is unsatisfiable first, then reduces the redundant axioms from the subset.

Based on those detected subsets for all unsatisfiable concepts, we can find minimal subsets of the ontology O which leads to all unsatisfiable concepts[13]. That can be used for knowledge workers to repair the ontology to avoid all unsatisfiable concepts.

[5] Stating that employees cannot be students.

[6] In order to do so, we extend the directly structral connection relation on concept sets, so that we can say something like an axiom β is connected with a concept c, i.e., $connected(\beta, C)$. It is easy to see that it does not change the definition.

Algorithm 2. Localize a minimal subset of O in which a concept C is unsatisfiable

$\Omega := \emptyset$
repeat
  for all $\beta_1 \in O \setminus \Omega$ **do**
    if there is a $\beta_2 \in \Omega$ such that $connected(\beta_1, \beta_2)$ or $connected(\beta_1, c)$ **then**
      $\Omega := \Omega \cup \{\beta_1\}$
    end if
  end for
until c is unsatisfiable in Ω
for all $\beta \in \Omega$ **do**
  if c is unsatisfiable in $\Omega - \{\beta\}$ **then**
    $\Omega := \Omega - \{\beta\}$
  end if
end for

Resolving Inconsistency Once the source of an inconsistency has been found, the conflict between the identified set of axioms has be to resolved. This task again is difficult, because in most cases there is no unique way of resolving a conflict but a set of alternatives. Often, there are no logical criteria selecting the best resolution. A common approach is to let the user resolve the conflict after it has been located.

Example 2. We again use the running example introduced in Example 1. Assume that we start out with the inconsistent ontology $O_3 = \{Employee \sqsubseteq Person, Student \sqsubseteq Person, PhDStudent \sqsubseteq Student, Employee \sqsubseteq \neg Student, PhDStudent \sqsubseteq Employee, PhDStudent(peter)\}$.

In this example the concept $PhDStudent$ is unsatisfiable. Starting with this unsatisfiable concept the algorithm finds the connected set $O_{31} = \{PhDStudent \sqsubseteq Student, PhDStudent \sqsubseteq Employee, PhDStudent(peter)\}$. The concept $PhDStudent$ is still safisfiable in O_{31}. Extending O_{31} with the connection relation the algorithm gets O_3. Reducing the redundant axioms, the algorithm finds the set $O_{32} = \{PhDStudent \sqsubseteq Student, Employee \sqsubseteq \neg Student, PhDStudent \sqsubseteq Employee\}$. Since $PhdStudent$ is the only unsatisfiable concept in this example, the knowledge workers can focus on the set O_{32} to repair O_3.

The approach proposed in this subsection is similar those in diagnosis[12]. There is a relativly well studied method for diagnosis, with a straightforward definitions: diagnosis is the smallest set of axioms that need to be removed to make the ontology consistent. These diagnoses can be calculated relatively easily on the basis of the minimal inconsistent subontologies. So, this covers the two parts of localizing and repairing inconsistencies (repairing an incoherent model by removing the minimal diagnoses).

4.3 Reasoning with Inconsistent Ontologies

In some cases it is unavoidable to live with inconsistencies, if consistency cannot be guaranteed and inconsistencies cannot be repaired. Nevertheless, there is still a need to reason about ontologies in order to support information access and integration of new information. We can summarize the task of reasoning with inconsistent ontologies: For a possibly inconsistent ontology O and a query q, the task of inconsistency reasoning is to return a meaningful query answer.

As shown above, the standard entailment is explosive, namely, any formula is a logical consequence of an inconsistent ontology. Therefore, conclusions drawn from an inconsistent ontology by classical inference may be completely meaningless. For an inconsistency reasoner it is expected that is able to return meaningful answers to queries, given an inconsistent ontology. In the case of a consistent ontology O, classical reasoning is sound, i.e., a formula ϕ deduced from O holds in every model of O. This definition is not preferable for an inconsistent ontology O as every formula is a consequence of O using a standard entailment $\models$. However, often only a small part of O has been incorrectly constructed or modelled, while the remainder of O is correct. Therefore, we propose the following definition of meaningfulness:

Definition 12 (Meaningfulness). *A query answer to a query $O \mid\!\approx \alpha?$ is meaningful iff the following two conditions are satisfied:*

1. soundness: *the answer is a consequence of a consistent subontology of O under the standard entailment $\models$,*
2. consistency: *the answer is a consistent query answer under the entailment $\mid\!\approx$.*

Algorithm 3. Linear extension strategy for the evaluation of query $O \mid\!\approx \alpha$

$\Omega := \emptyset$
repeat
  $\Omega' := \{\beta_1 \in O \setminus \Omega$: there exists a $\beta_2 \in \Omega \cup \{\alpha\}$ such that $connected(\beta_1, \beta_2)\}$
  if $\Omega' = \emptyset$ **then**
    return $O \not\mid\!\approx \alpha$
  end if
  $\Omega := \Omega \cup \Omega'$
  if Ω inconsistent **then**
    $\Omega'' := maximal\_consistent\_subontology(\Omega)$
    if $\Omega'' \models \alpha$ **then**
      return $O \mid\!\approx \alpha$
      else return $O \not\mid\!\approx \alpha$
    end if
  end if
until $\Omega \models \alpha$
return $O \mid\!\approx \alpha$

The general strategy for processing inconsistent ontologies is: given a connection/relevance relation (c.f. Definition 10), we select some consistent subontology from an inconsistent ontology. Then we apply standard reasoning on the selected subontology to find meaningful answers. If a satisfying answer cannot be found, the relevance degree of the selection function is made less restrictive thereby extending the consistent subontology for further reasoning. If an inconsistent subset is selected, we call the over-determined processing(ODP)[8]. One of the ODP strategies is to find the set of the maximal consistent subontologies of the selected set. If there exist contradictory answers from those maximal consistent subontologies, the algorithm will return 'unknown'. A linear extension strategy with an ODP for the evaluation of a query '$O \mid\!\approx \alpha?$' is described in Algorithm 3. We can prove the following property[8]:

Proposition 2 (Meaningfulness of Linear Extension Strategy). *The answers which are obtained by the linear extension strategy are meaningful.*

Example 3. Consider the inconsistent ontology $O_3 = \{Employee \sqsubseteq Person, Student \sqsubseteq Person, PhDStudent \sqsubseteq Student, PhDStudent \sqsubseteq Employee, Employee \sqsubseteq \neg Student, PhDStudent(peter)\}$.

Assume now we wanted to ask the query $O_3 \mid\!\approx Student(peter)$?. Using standard entailment we would obtain no meaningful answer, as both $Student(peter)$ and $\neg Student(peter)$ are entailed by the ontology. By the linear extension on the connection relation with $Student(peter)$, the algorithm will construct the ontology $\Omega = \{PhDStudent(peter), PhDStudent \sqsubseteq Employee, PhDStudent \sqsubseteq Student\}$. This ontology Ω is consistent, and $\Omega \models \alpha$. Thus, the algorithm concludes that $O_3 \mid\!\approx Student(peter)$.

4.4 Multi-version Reasoning

Multi-version reasoning is an approach that tries to cope with possible inconsistencies in changing ontologies by considering not only the latest version of an ontology, but all previous versions as well. This approach mostly applies in cases where the problem is not so much an inconsistency in the ontology itself, but inconsistencies that arise from the interaction of the ontology with its environment in terms of instance data and applications. We consider the sequence of ontologies $O_1 \prec \cdots \prec O_n$ where the ordering relation is defined as:

$$O_i \prec O_j \Leftrightarrow \exists \text{oco}_{composite} : \text{oco}_{composite}(O_i) = O_j$$

Intuitively, O_n is the current version of the ontology. $O_1, \cdots, O_{n-1}$ are older versions of the same ontology that have been created from the respective previous ontology in terms of a composite change action. We can assume that each of the ontologies is consistent. Further, we assume that an application expresses its requirements for compatibility as an expectation α, for which there is an ontology O_i in the sequence such that $O_i \cup \{\alpha\}$ is consistent.

Based on these assumptions, the task of ensuring consistency reduces to the task of finding the right version O_i of the ontology in the sequence of versions. This task requires the ability to determine the satisfiability of certain expressions across the different versions of the ontology. This can be done using an extension of the ontology language called $\mathcal{L}+$ with the operator $\mathbf{PreviousVersion}\phi$, which is read as '$\phi$ holds in the previous version', the operator $\mathbf{AllPriorVersions}\phi$, which is read as '$\phi$ holds in all prior versions', and the operator $\mathbf{SomePriorVersion}\phi$, which is read as '$\phi$ holds in some prior versions'.

Using these basic operators, we can define a rich set of query operators for asking specific questions about specific versions of the ontology and relations between them. In the case where $O_n \cup \{\alpha\}$ is inconsistent, we can for example check whether the previous version can be used ($\mathbf{PreviousVersion}\,\alpha$) and whether there is a version at all that can be used instead ($\mathbf{SomePriorVersion}\,\alpha$). For the formal semantics of these operators we refer the reader to [7].

Example 4. Consider we have an ordered relation of ontologies $O_1 \prec O_2$, using the ontologies from Example 1. Now assume a compatibility criteria that has to fulfilled

for compatibility: $\alpha = Employee(peter)$, i.e. a knowledge base in which Peter is an employee. The latest version O_2 is compatible with the compatibility criteria α as $O_2 \cup \{\alpha\}$ is consistent. However, O_1 does not meet the compatibility requirements, as $O_1 \cup \{\alpha\}$ is inconsistent (It still contained the axiom stating the disjointness of students and employees). In fact, it holds that $\mathbf{AllPriorVersions}\, \neg Employee(peter)$.

5 Comparison and Evaluation

We are going to compare the four approaches dealing with inconsistency, and make an evaluation on them. By the evaluation, we want to suggest several guidelines for system developers to know under which circumstance which approach is more appropriate.

5.1 Different Functionality

A first major difference that is revealed by the formal analysis in the previous section is the fact that the different methods for dealing with inconsistent ontologies actually have very different functionality (their input/output-relations are rather different). Consequently, they solve rather different tasks, and are suited for different use-cases. The situation is summarised in Table 1.

Table 1. Comparison of Approaches

Approach	Applied At	Input	Output
Consistent Evolution	Development	Consistent Ontology, Change	Consistent Ontology
Inconsistency Repair	Development	Inconsistent Ontology	Consistent Ontology
Inconsistency Reasoning	Runtime	Consistent Ontology, Query	Meaningful answer
Multi-version reasoning	Runtime	Versions of Ontologies, Query	Consistent Answer

Dependence on query. First, this table shows that two of the methods depend on which user-query is given to the ontology (reasoning with inconsistency and multi-version reasoning). Consequently, these two methods are only applicable at *runtime*, when a user interacts with the ontology. The other two methods (ontology evolution and inconsistency repair) are independent of user-queries, and can thus already be applied at ontology *development time*.

Known or unknown change. The two methods that are applicable at ontology development time are actually very similar (as is apparent from sections 4.1 and 4.2). A crucial difference is that the first of these (ontology evolution) requires knowledge of the change that caused the ontology to become inconsistent: algorithm 1 requires the change α to be known, which is not the case with 2. This is clearly a restriction on the applicability of ontology evolution, which comes in exchange for the benefit of a simpler algorithm.

Known or unknown history. The two query-dependent approaches also differ in their respective input-requirements: multi-version reasoning requires a *history* of ontology-versions to be available, which is a very strong demand, often not feasible in many settings, in particular in combination with its runtime usage.

5.2 Other Aspects

Heuristics. Another difference between the various approaches is the extent to which they employ heuristics: in reasoning with inconsistency, one heuristically chooses a consistent subontology that is good enough to answer the query (it need not be minimal, just small enough to be consistent, and large enough to answer the query). In contract, both Evolution and Repair aim at the *smallest* impact on the inconsistent ontology.

Efficiency. Finally, one would expect the various approaches to differ drastically in their computational efficiency. Some observations can be made immediately: the Evolutionary approach exploits the knowledge about the cause of the inconsistency, and can therefore be more efficient then Repair, which does not have access to this information. However, the cost of all of the algorithms described in this paper are dominated by untractable operations such as checking the unsatisfiability of a concept or the inconsistency of an entire ontology. Consequently, worst-case complexity analysis is not going to tell us anything interesting here. Instead, work will have to be done on average-case complexity analysis and experiments with realistic datasets to gain more insight into the relevative costs of each of the approaches.

Knowledge Requirements. Finally, the approaches differ in the knowledge that is required to operate them:

- the repair approach requires the ontology developers to have sufficient domain knowledge to decide which part of the ontology should be removed to recover consistency. On the other hand, once done, it needs no additional expertise from the ontology users.
- Reasoning with inconsistencies on the other hand emposes no knowledge requirements on the developers, but requires some (weak) knowledge from the users to determine whether a query answer is acceptable.
- Ontology versioning places again a heavy knowledge requirement on the user in order to decide which version is most suitable for their application.

6 Related Work

The evolution of ontologies has been addressed by different researchers by defining change operations and change representations for ontology languages. Change operations have been proposed for specific ontology languages. In particular change operations have been defined for OKBC, OWL [9] and for the KAON ontology language [14]. All approaches distinguish between atomic and complex changes. Different ways of representing ontological changes have been proposed: besides the obvious representation as a change log that contains a sequence of operations, authors have proposed to represent changes in terms of mappings between two versions of an ontology [11].

The problem of preserving integrity in the case of changes is also present for ontology evolution. On the one hand the problem is harder here as ontologies are often encoded using a logical language where changes can quickly lead to logical inconsistency that cannot directly be determined by looking at the change operation. On the other hand, there are logical reasoners that can be used to detect inconsistencies both within the ontology and with respect to instance data. As this kind of reasoning is often

costly, heuristic approaches for determining inconsistencies have been proposed [9,15]. While deciding whether an ontology is consistent or not can easily be done using existing technologies, repairing inconsistencies in ontologies is an open problem although there is some preliminary work on diagnosing the reasons for an inconsistency which is prerequisite for a successful repair [13].

The problem of compatibility with applications that use an ontology has received little attention so far. The problem is that the impact of a change in the ontology on the function of the system is hard to predict and strongly depends on the application that uses the ontology. Part of the problem is the fact that ontologies are often not just used as a fixed structure but as the basis for deductive reasoning. The functionality of the system often depends on the result of this deduction process and unwanted behavior can occur as a result of changes in the ontology. Some attempts have been made to characterize change and evolution multiple versions on a semantic level [3,4]. This work provides the basis for analyzing compatibility which currently is an open problem.

7 Conclusion

Unlike work in traditional knowledge engineering, knowledge intensive applications on the Web will not be able to ignore the issue of inconsistent knowledge in general, and of inconsistent ontologies in particular. This has been recognised in various contributions to the literature that propose different ways of dealing with inconsistent ontologies. These approaches differ both in the machinery they use, and in the way they propose to deal with inconsistent ontologies, ranging from avoiding inconsistencies, to diagnosing and repairing the inconsistencies, to trying to reason in the presence of the inconsistencies, and to tracking the inconsistencies over the development history of an ontology.

In this paper, we have rephrased four existing approaches to dealing with inconsistent ontologies in terms of a set of elementary definitions. This allowed us to compare these rather different approaches on an equal footing. This comparison revealed among other things that what originally seemed to be different approaches to the same problem (namely dealing with inconsistent ontologies) are actually solutions that apply in very different settings: at ontology-development time or at ontology-use time, and requiring different pieces of information (the cause of the inconsistency, or the history of the ontology changes). For the respective approaches, we provide implementations, which are available at `http://www.aifb.uni-karlsruhe.de/WBS/pha/owlevolution/`.

Acknowledgements. Research reported in this paper has been partially financed by EU in the IST projects SEKT (EU IST-2003-506826) and Knowledge Web (EU IST-2003-507482).

References

1. Giorgos Flouris. Belief change in arbitrary logics. In *HDMS*, 2004.
2. P. Haase and L. Stojanovic. Consistent evolution of OWL ontologies. In *Proceedings of the Second European Semantic Web Conference, Heraklion, Greece, 2005*, MAY 2005.
3. J. Heflin. *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment*. Phd thesis, University of Maryland, 2001.

4. J. Heflin and J. Z. Pan. A model theoretic semantics for ontology versioning. In *Third International Semantic Web Conference*, pages 62–76, Hiroshima, Japan, 2004. Springer.
5. I. Horrocks and P. F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. *Journal of Web Semantics*, 1(4), 2004.
6. I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.
7. Z. Huang and H. Stuckenschmidt. Reasoning with multiversion ontologies: a temporal logic approach. In *Proceedings of the 2005 International Semantic Web Conference (ISWC'05)*, 2005.
8. Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI'05)*, pages 254–259, 2005.
9. M. Klein. *Change Management for Distributed Ontologies*. Phd thesis, Vrije Universiteit Amsterdam, 2004.
10. D. McGuinness and F. van Harmelen. OWL Web Ontology Language. Recommendation, W3C, 2004. http://www.w3.org/TR/owl-features/.
11. N.F. Noy and M.A. Musen. The prompt suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.
12. R. Reiter. A theory of diagnosis from first principles. *Artif. Intelligence*, 32(1):57–95, 1987.
13. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI'03*, Acapulco, Mexico, 2003. Morgan Kaufmann.
14. L. Stojanovic. *Methods and Tools for Ontology Evolution*. Phd thesis, University of Karlsruhe, 2004.
15. H. Stuckenschmidt and M. Klein. Integrity and change in modular ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI'03*, Acapulco, Mexico, 2003. Morgan Kaufmann.