Solving Terminological Inconsistency Problems in Ontology Design

Ekaterina Ovchinnikova, Tonio Wandmacher, Kai-Uwe Kühnberger
Institute of Cognitive Science
University of Osnabrück
Osnabrück, Germany
e.ovchinnikova@uos.de, tonio.wandmacher@uos.de,
kkuehnbe@uos.de

Abstract: Information models and ontologies, although originally developed for different applications and most often used in different disciplines, share several common features. It is natural to assume that techniques applicable for knowledge representation tasks based on ontologies, can be used for information models as well. In this paper, the focus will be on resolving inconsistencies in ontology design. In particular, an algorithmic solution is proposed that allows to automatically rewrite certain types of occurring inconsistencies in terminological hierarchies. Furthermore an experimental evaluation of the proposed algorithm is sketched.

Introduction

Information models play an important role in information systems, current state of the art tools for management tasks, and the controlling of production processes. The overall aim of information models is primarily to structure management information.[1] Although there is a variety of these models [M01] making it difficult to keep track of the different versions and their applications, it seems to be the case that most of these models contain a core of certain key features. Some examples of such features are summarized in the following list (cf. [W03] for the common information model):

- A hierarchical structure is imposed on information types.
- A conceptual (often object-orientied) perspective allows the ordering of entities into instances, properties, classes, subclasses, operations, and relations.
- Additional information can be coded by meta schemes.

Most of the mentioned aspects are in one or the other form also contained in classical ontology-based frameworks for knowledge engineering tasks originally developed for other purposes like semantic web applications, expert systems, or text processing applications. For example, ontologies are based on classes, instances, a subsumption relation (i.e. a hierarchical structure of classes), relations between classes etc. In a certain sense, information models can be considered as a special type of ontologies [DVBAPD04]. Therefore it is not surprising that there is the possibility to map concepts of information models to constructs of an

[1] Distributed Management Task Force: CIM Tutorial, available online at: http://www.wbemsolutions.com/tutorials/CIM/.

approrpiate ontological language: for example, in [QAWBS04], the authors propose such a mapping for CIM and a RDF/S ontology.

A further corresponding aspect is the hierarchical set-up of the usability of both concepts. Whereas general and reusable ontologies build the basis of this hierarchy and more specific and usable domain ontologies for concrete applications are located at the top, this is mirrored by certain information models as well. In [DVBAPD04], the authors propose, for example, to associate application ontologies with extension schemas of CIM, domain ontologies with the CIM common model or generic domain ontologies with the CIM core mode. Such correspondences do not only hold for CIM, but on a more general level as well. For example, using a framework from software development, one can also find similar correspondences between MDA layers (Model-Driven Architecture) [GDD05] and ontologies: whereas UML based models correspond to generic domain ontologies, meta-object families (MOF) in MDA can be associated to representation ontologies. Last but not least, an important fact, demonstrating the applicability of ontologies to information modeling is the fact that formal ontologies are also used as a conceptual (or terminological) component in diverse information systems (see [G98] for an overview).

Although there are several similarities between information models and ontologies, there is also an important differnce, namely with respect to the usage of reasoning techniques: on the one hand, with respect to ontologies reasoning techniques can be applied in order to deduce new facts, because ontological knowledge is based on axiomatic specifications defined in precise logical formalisms. In a certain sense, ontologies were developed in order to allow the implementation of efficient reasoning techniques. On the other hand, in information models, reasoning applications at least do not play a similarly important role (if used at all), although newer developments attempt to extend information models towards this direction [AFFS06].

A well-known problem in knowledge engineering are occurring inconsistencies. In information models, the situation is quite similar: because analysis tools for information models require verification and validation techniques occurring inconsistencies need to be resolved – if possible automatically [M98]. In this paper, we will suggest and discuss an approach to automatically resolve inconsistencies in hierarchically structured terminological knowledge bases.

Ontologies and Description Logics

Although there is no generally accepted definition of what an ontology is [SM01], from an abstract point of view, an ontology contains as a core terminological knowledge in form of hierarchically structured concepts. These concepts can be enriched by relations specifying constraints on them.

Certain standards allow to represent ontological knowledge in well-defined languages. In recent years the fast development of the world-wide-web has brought about a wide variety of standards for knowledge representation. Probably the most

important existing markup language for ontology design is the Web Ontology Language *OWL* in its three different versions: *OWL Lite*, *OWL DL*, and *OWL Full* [OWL04]. The mentioned OWL versions are hierarchically ordered, such that *OWL Full* includes *OWL DL*, and *OWL DL* includes *OWL Lite*. Consequently they differ in their expressive power with respect to possible concept formations.

All versions of OWL are based on the logical formalism called description logic (DL) [BCMNP03]. Description logics were originally designed for the representation of terminological knowledge and reasoning processes. They can be characterized as subsystems of first-order predicate logic using at most two variables. Two points should be mentioned:

- In comparison to full first-order logic, description logics are – due to their restrictions concerning quantification – rather weak logics with respect to their expressive power.
- DLs can be used to characterize the different OWL versions. For example, *OWL DL* can be logically characterized as a syntactic variant of the description logic *SHOIN*(**D**) [MSS04].

A classical distinction in description logic is to separate terminological knowledge about concepts and facts in two different data structures. Knowledge about the hierarchical structure of concepts is coded in the so-called TBox (terminological box) whereas knowledge about facts is coded in the ABox (assertion box).

A DL terminology contains terminological axioms that define concepts occuring in the domain of interest. Core axioms are of the form $A \sqsubseteq D$ *(*meaning that *A* is a subconcept of *D)* or $A_1 \equiv A_2$ (meaning that concepts A_1 and A_2 are logically equivalent), where *A* stands for a concept name and *D* stands for a concept definition constructed from concept and role names with the help of syntactic rules using classical logical operators. An ABox is a finite set of facts *C*(*a*) or *R*(*b,c*) where *C* is a concept name, *R* is a relation name, and *a, b* and *c* are individuals. *C(a)* means that an individual *a* belongs to a concept *C* and *R(b,c)* means that individuals *b* and *c* are connected with a relation *R*.

DL expressions are interpreted in a classical model theoretic way: formally, an interpretation *I* is a mapping assigning to each concept name *A* a subset of the domain Δ and to each role name *R* a subset of the Cartesian product $\Delta \times \Delta$. An interpretation *I* is a model of a Tbox *T* if for every inclusion axiom $A \sqsubseteq D$ we have $I(A) \subseteq I(D)$ and for every equality axiom $A_1 \equiv A_2$ it holds: $I(A_1) = I(A_2)$. A concept name *A* is satisfiable towards *T* if there is a model *I* of *T* such that *I*(*A*) is non-empty.

Inconsistency in Information Modeling

Since inaccurately formulated information models may cause applications to work incorrectly, the problem of consistency in information modeling is widely discussed

in the literature (cf. [M98] for an overview). Consider an example of a logical inconsistent information model represented in natural language (taken from [L81]).

Example 1.
Everybody who has an income is a shareholder. No shareholder is also an employee. Every employee has an income.

The informal description in Example 1 is inconsistent, since it is impossible to find an employee who will satisfy this model, because according to the model every employee has an income, is a shareholder, and is not a shareholder simultaneously. Thus, the model in Example 1 is inconsistent and is unusable in practice.

In order to ascertain that an information model is meaningful and useful one has to perform different types of checking. An important type of checking that is usually executed with the help of automatic verification tools (for example [E05, BKSL01]) concerns formally defined models. It can be formally verified whether such models satisfy syntactic and semantic rules of the corresponding representation language. Several approaches to the verification of formal information models use logical mechanisms for the detection and elimination of logical contradictions (cf. [BKSL01, L81, SSJM04]).

High emphasis is placed on the verification of the consistency of UML[2] models ([SSJM04, E05]). In particular, Simmonds et al. [SSJM04] show that UML models can be expressed in a description logic and, thus, DL reasoners can be used to detect some types of inconsistencies in UML information models. Furthermore newer developments in information modeling for management tasks enrich information models with reasoning capabilities: in [AFFS06], the authors show that a certain description logic is appropriate to capture the semantics of CIM models.[3] In case a calculus for reasoning is available, the question whether the underlying information model is consistent becomes even more important, due to the fact that reasoners are not very robust with respect to inconsistent data. If inconsistencies do occur, it is desirable to automatically resolve their occurrence.

A simple reason for occurring inconsistencies in information models may be an ontology that represents a terminological component of the model. If this component is inconsistent, the proper working of the whole systems is endangered. In the following sections, we describe an algorithm that resolves inconsistencies in terminological knowledge bases.

[2] http://www.uml.org

[3] Technically it is shown that the DL logic $AL\varepsilon CNOQ^{-}_{HR+\circ}$ captures the semantics of CIM. This logic contains constructors for atomic, empty, and domain concepts. Furthermore the usual logical connectives are covered, as well as role constructors for the various versions of quantification and cardinality restrictions. Last but not least, constructors for inverse role, transitive roles, and role composition are available. The described DL is rather expressive.

Terminological Inconsistencies

The notion of *terminological inconsistency* has several meanings. In [HS05], for example, three types of inconsistency are distinguished:

- *Structural inconsistency* is defined with respect to the underlying representation language. A knowledge base is structurally inconsistent, if it contains axioms violating the syntactical rules of the representation language (for example, OWL *DL*).
- *Logical inconsistency* is defined on the basis of formal semantics of the knowledge base. An ontology is logically inconsistent, if the ontology has no model.
- *User-defined inconsistency* is related to application context constraints defined by the user.

In this paper, we consider logical inconsistency only. In particular, the main focus lies on contradicting unsatisfiable terminologies.

Definition 1. *A terminology T is unsatisfiable if there exists a concept C that is defined in T and is unsatisfiable.*

Informally, Definition 1 implies that an inconsistent ontology necessarily contains logical contradictions. An ontology can be inconsistent only if its underlying logic allows negation. Ontologies share this property with every logical system (like, for example, first-order logic). In practice, logical inconsistency can be caused by several reasons. For example, errors in the automatic ontology learning procedure or mistakes of the ontology engineer can generate unintended contradictions.

Another type of logical inconsistency is connected with polysemy. If an ontology is learned automatically, then it is hardly possible to distinguish between senses of words that represent concepts in texts. Suppose, the concept *tree* is declared to be a subconcept both of *plant* and of *data structure* (where *plant* and *data structure* are disjoint concepts). These two interpretations of *tree* are true, but it is still necessary to describe in the ontology two different concepts with two different identifiers (e.g. *TreePlant*, *TreeStructure*).

Finally, there is a set of problems related to generalization mistakes. Let us consider an example. Suppose that the ontology contains the following facts:

Example 2.

$\texttt{Bird} \sqsubseteq \texttt{CanFly}$	(*Birds are creatures that can fly.*)
$\texttt{CanFly} \sqsubseteq \texttt{CanMove}$	(*If a creature can fly then it can move.*)
$\texttt{Canary} \sqsubseteq \texttt{Bird}$	(*Canary is a bird.*)
$\texttt{Penguin} \sqsubseteq \texttt{Bird} \sqcap \neg\texttt{CanFly}$	(*Penguin is a bird and cannot fly.*)

In Example 2 the statement *birds can fly* is too general. After an exception (penguin) appears, the ontology becomes inconsistent, since penguin is declared to

be a bird, but it cannot fly. It is easy to see that the inconsistency problem in Example 1 can be expressen in DL and considered as a terminological inconsistency.

Related Work

Several approaches were proposed to treat inconsistencies in ontology design. Three major types can be distinguished:

- Approaches identifying inconsistencies but not resolving them.
- Approaches that try to reason with inconsistent ontologies.
- Approaches that (semi-)automatically resolve inconsistencies.

Concerning the first type of solutions, an interesting proposal can be found in [GLW06]. The authors propose a non-conservative extension of ontologies in the case a concept description is satisfiable prior to an extension and unsatisfiable afterwards: A witness concept description is introduced which reports the inconsistency to the knowledge engineer. A number of approaches automatically detect sets of axioms that are responsible for a particular inconsistency (cf. [WHRDS05], [SC03]). Although these accounts cannot automatically resolve existing inconsistencies, they can help the knowledge engineer to identify occurring problems.

The second type of solutions contains approaches that use several well-known techniques from non-monotonic reasoning, like default sets [HV02], planning systems [BLMSW05], or epistemic operators [KP05]. Unfortunately these approaches go beyond the expressive power of description logics and cannot be represented in a description logic framework.

Finally, the third type of solutions comprises approaches that (semi)-automatically resolve inconsistencies by removing ([FFIPS04], [HS05], [HHHSS05], [K06]) or rewriting ([LPSV06], [OK06]) problematic axioms or parts of axioms. In general, removing problematic information can cause a loss of intended entailments. [HS05], [HHHSS05], and [K06] suggest to use different kinds of ratings that can help to detect the least-damage removal of axioms. [K06] also applies a set of error patterns to problematic axioms: If an axiom matches to such patterns, then it is rewritten according to the corresponding repair pattern. [LPSV06] extend the tableau-based algorithm in order to find sets of axioms causing inconsistency and the set of "helpful" changes that can be performed to debug the ontology. [OK06] propose an automatic amalgamation procedure changing the original ALE-ontology[4], if it conflicts with new information and rewriting overgeneralized concept definitions.

[4] See [BCMNP03] for the definition of the *ALE* description logic.

Adaptive Ontologies

In this section, we informally describe an approach to resolve inconsistent ontologies that is based on the ideas technically introduced in [OK06] and developed in [OK07]. The mentioned approach is extended by the treatment of polysemy problems. Given an inconsistent ontology we want to change it automatically in order to obtain a consistent one, according to the following principles:

- The performed changes have to be relevant and intuitive.
- The changed ontology is formalized in a description logic language.
- As few pieces of information as possible are removed from the ontology.

In general accidental mistakes cannot be fixed automatically. But the polysemy problem can be resolved by renaming concepts with polysemous names. Furthermore overgeneralized concepts can be redefined so that problematic pieces of information will be deleted from their definitions.

Adaptation Algorithm

The proposed approach treats inconsistent ontologies or consistent ones that are extended with additional axioms conflicting with the original knowledge base. Given a consistent ontology O (possibly empty) the procedure adds a new axiom A to O. If $O^+ = O \cup \{A\}$ is inconsistent then the procedure tries to find a polysemy or an overgeneralization and repairs O^+ .

For the sake of simplicity let us restrict ourselves to the description of the adaptation procedure for the TBox presuming a similar treatment for the ABox instantiations. Suppose that the new axiom A represents a definition of a concept C. Regarding the TBox, O^+ is inconsistent if a subconcept C' of the newly introduced or newly defined concept C is unsatisfiable.

Unfortunately, it is impossible to distinguish between accidental mistakes, polysemy problem and overgeneralization strictly logically. Our algorithm inspects the definitions of the unsatisfiable concept C', tries to fish out overgeneralized concepts subsuming C' and regeneralize these concepts. If no overgeneralized concepts have been found, then the algorithm defines which concepts are suspected to be polysemous and renames these concepts (by default or given the consent of the user).

This algorithm can be used for (a) resolving inconsistencies in an ontology, (b) adapting a consistent base ontology O to new axioms, and (c) merging ontologies (in this case there are two consistent ontologies given, but their union can become inconsistent).

Regeneralization of Overgeneralized Concepts

We will illustrate the regeneralization of the overgeneralized concepts on the ontology in Example 2. Since the definition of the concept *Bird* is overgeneralized, it needs to be rewritten. We wish to retain as much information as possible in the ontology. The following solution is proposed:

Adapted ontology from Example 2.

$\texttt{Bird} \sqsubseteq \texttt{CanMove}$	(*Birds are creatures that can move.*)
$\texttt{FlyingBird} \sqsubseteq \texttt{Bird} \sqcap \texttt{CanFly}$	(*Flying birds are birds that can fly.*)
$\texttt{CanFly} \sqsubseteq \texttt{CanMove}$	(*If a creature can fly then it can move.*)
$\texttt{Canary} \sqsubseteq \texttt{FlyingBird}$	(*Canary is a flying bird.*)
$\texttt{Penguin} \sqsubseteq \texttt{Bird} \sqcap \neg\texttt{CanFly}$	(*Penguin is a bird and cannot fly.*)

We want to keep in the definition of the concept *Bird* (subsuming the unsatisfiable concept *Penguin*) a maximum of information that does not conflict with the definition of *Penguin*. The conflicting information is moved to the definition of the new concept *Flying bird*, which is declared to subsume all former subconcepts of *Bird* (such as *Canary* for example).

The example below represents a case where two overgeneralized definitions of the same concept conflict with each other.

Example 3.

$\texttt{Child} \sqsubseteq \forall\texttt{likes.Icecream}$	(*Children like only icecream.*)
$\texttt{Icecream} \sqsubseteq \texttt{Sweetie}$	(*Icecream is a sweetie.*)
$\texttt{Chocolate} \sqsubseteq \texttt{Sweetie}$	(*Chocolate is a sweetie.*)
$\texttt{Icecream} \sqsubseteq \neg\texttt{Chocolate}$	(*Icecream and chocolate are disjoint concepts.*)
$\texttt{Child} \sqsubseteq \forall\texttt{likes.Chocolate}$	(*Children like only chocolate.*)

In Example 3, the definitions of *Child* (*Children like only icecream* and *Children like only chocolate*) are too specific. *Icecream* and *Chocolate* being disjoint concepts produce a conflict.
It seems to be an intuitive solution to replace these concepts by their least common subsumer (see [CBH93]) *Sweetie*. Furthermore it is plausible to claim that children like only sweeties without specifying it precisely, as described below:

Adapted ontology from Example 3.

$\texttt{Child} \sqsubseteq \forall\texttt{likes.Sweetie}$	*(Children like only sweeties.)*
$\texttt{Icecream} \sqsubseteq \texttt{Sweetie}$	(*Icecream is a sweetie.*)
$\texttt{Chocolate} \sqsubseteq \texttt{Sweetie}$	(*Chocolate is a sweetie.*)
$\texttt{Icecream} \sqsubseteq \neg\texttt{Chocolate}$	(*Icecream and chocolate are disjoint concepts.*)

The natural question is: how to detect overgeneralized concepts? Let us describe the regeneralization procedure avoiding formal aspects (see [OK06] for more detailes). If an unsatisfiable concept *X* is defined in the TBox *T* by the definitions *A*

and B[5] that are logicaly conflicting (their conjunction is unsatisfiable), then the following options can be distinguished:

- A and B are disjoint concept descriptions having common subsumers (Example 3):
  The solution in this case is to replace the definitions A and B of X with their least common subsumer.
- A is defined in T and some definition D_A of A conflicts with B (Example 2):
  This case is considered as the overgeneralization of A. The definition D_A has to be revised as follows: (a) D_A is replaced with its minimal specific superdescription that does not conflict with B; (b) a new concept A' is added to the TBox as a subconcept of A and D_A; (c) A is replaced with A' in the definitions of all its subconcepts except in the definition of X.
- A and B are defined in T, a definition D_A of A conflicts with B, and a definition D_B of B conflicts with A:
  In this case there is no unique solution. On the one hand the concept X is suspected to be polysemous. Here, the preferred solution is to split the definition of X and rename X as, for example, X_1 and X_2. On the other hand we may face two overgeneralized concepts, one or both definitions of which can be changed in the way described in the previous option (2). By default the procedure considers X to be polysemous. But if the ontology engineer decides to supervise the procedure in order to avoid possible mistakes, she can consider all such ambiguous cases and choose a proper solution.
- Otherwise:
  The concept X is suspected to be polysemous as in the previous option.

System Architecture

The overall architecture of the system is depicted in Figure 1. A base ontology O is given and updated by new axioms A that are extracted automatically with the help of some external tool or added manually by an ontology engineer. An integration engine checks the resulting ontology for consistency. In the case inconsistencies occur, the proposed procedure can be used to resolve these inconsistencies. The result is an integrated consistent ontology O'. The whole process can be considered as a cycle: the newly computed ontology O' can be updated by new axioms and resolved in the next cycle.

[5] The definitions of X are previously converted to conjunctive normal form and split, such that every conjunction is divided into two: $\{X \sqsubseteq D_1 \sqcap D_2\} \rightarrow \{X \sqsubseteq D_1, X \sqsubseteq D_2\}$, $\{X \equiv D_1 \sqcap D_2\} \rightarrow \{X \sqsubseteq D_1, X \sqsubseteq D_2, D_1 \sqcap D_2 \sqsubseteq X\}$. Thus, every definition of X is a (negated) atomic concept or relational restriction.

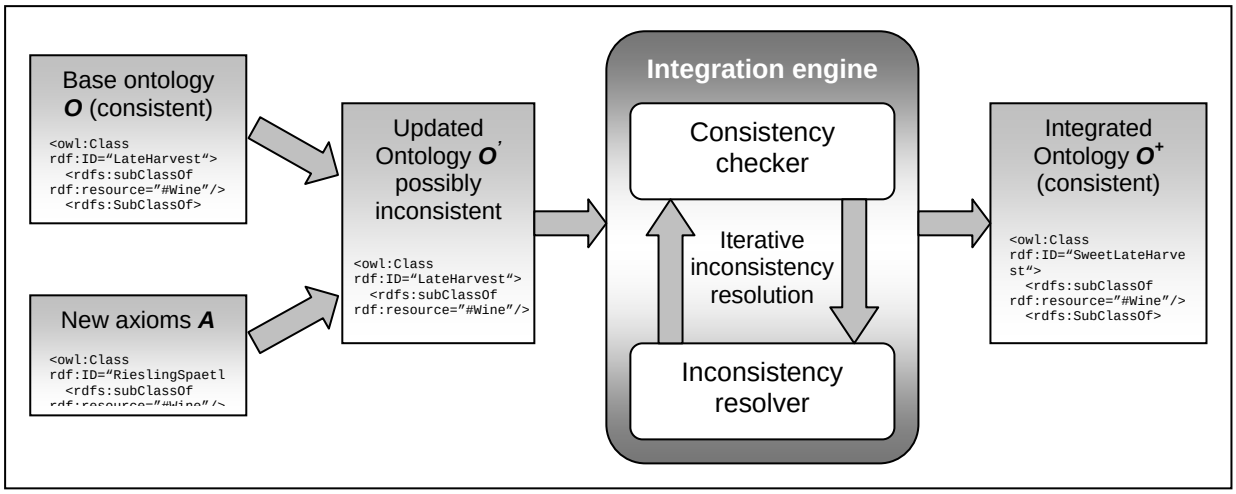

Figure 1. Integration process of a consistent base ontology O and a set of arriving new axioms A.

Thus, the core of the system consists of two modules: consistency checker and inconsistency resolver. The consistency checker returns a set of conflicts and sets of axioms that are responsible for each conflict. Using this information and the original terminology the inconsistency resolver rewrites a problematic axiom and conveys the changed ontology to the consistency checker again. The procedure terminates when there are no more contradictions in the ontology.

To make description logic inferences available for our algorithm we have integrated the KAON2 DL plug-in[6] to our system designed for managing of and reasoning on OWL ontologies. We use KAON2 as a parser for the OWL and RDF/XML representation languages and as an independent "black box" reasoner. At present the KAON2 reasoner fully supports OWL Lite with some extensions.

Experimental Evaluation

We roughly sketch in this section some practical experiments. The prototype implementation of the ideas presented here was designed for resolving inconsistencies in OWL Lite ontologies. As a base ontology we have taken the famous wine ontology[7] describing different sorts of wine, grapes and wine regions. This ontology was created manually and thus does not contain any inconsistencies.

In order to create a domain related corpus, we generated a document set which was automatically crawled from the web with the *BootCat Tools* [BB04], using the vocabulary of the wine ontology as seed terms. We thus obtained a domain corpus of 288 documents comprising 182,754 token. This corpus served as input to the ontology extraction step. For this purpose we decided to use the freely available *Text2Onto*[8] tool, developed at the *AIFB*, Karlsruhe, (Germany), because this tool is capable of extracting not only basic relations such as taxonomy, but also

6 http://kaon2.semanticweb.org

7 http://www.w3.org/TR/owl-guide/wine.owl

8 http://ontoware.org/projects/text2onto/. Special thanks to Johanna Völker, who gave us helpful support!

disjointness and equivalence (see [CV05]). In the hereby automatically generated ontology, we however found only concepts (2155), instances (986), subclass (385) and instance (211) relations. We then manually filtered the extracted relations to exclude errors. We also manually reformatted some relations to avoid a syntactic mismatch with the original ontology. This finally resulted in an ontology of 137 valid subclass relations and 83 instance relations. The automatically generated ontology proved to contain several logical inconsistencies with respect to the original wine ontology. Several cases of polysemy were detected as, for example, the *champagne* class being a subclass of both *region* and *wine* or the *pinot noir* class which was defined to be a subclass of *wine* and *grape*.

The algorithm discovered also several cases of real overgeneralization. Consider the following example:

O = {
```
<owl:Class rdf:ID="LateHarvest">
   <rdfs:subClassOf rdf:resource="#Wine"/>
   <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasSugar"/>
        <owl:allValuesFrom rdf:resource="#Sweet"/>
      </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Dry">
    <rdfs:subClassOf rdf:resource="#WineSugar"/>
    <owl:disjointWith rdf:resource="#Sweet"/>
 </owl:Class>
 <owl:Class rdf:ID="Sweet">
    <rdfs:subClassOf rdf:resource="#WineSugar"/>
    <owl:disjointWith rdf:resource="#Dry"/>
 </owl:Class>
}
```

O′ = {
```
<owl:Class rdf:ID="RieslingSpaetlese">
   <rdfs:subClassOf rdf:resource="#LateHarvest"/>
   <rdfs:subClassOf rdf:resource="#Riesling"/>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#hasSugar"/>
       <owl:allValuesFrom rdf:resource="#Dry"/>
     </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>
}
```

In this example[9], the concept *RieslingSpaetlese* represents an exception to the (overgeneralized) definition stating that every *late harvest* wine is a sweet wine. Our system changes the corresponding ontology fragment as follows:

[9] The example has slightly been modified. In the original *Text2Onto* output, *RieslingSpaetlese* is modeled as a subconcept of *DryWine*.

```
{
<owl:Class rdf:ID="LateHarvest">
   <rdfs:subClassOf rdf:resource="#Wine"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasSugar"/>
        <owl:allValuesFrom rdf:resource="#WineSugar"/>
      </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="SweetLateHarvest">
   <rdfs:subClassOf rdf:resource="#LateHarvest"/>
   <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasSugar"/>
        <owl:allValuesFrom rdf:resource="#Sweet"/>
      </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Dry">
    <rdfs:subClassOf rdf:resource="#WineSugar"/>
    <owl:disjointWith rdf:resource="#Sweet"/>
 </owl:Class>
 <owl:Class rdf:ID="Sweet">
    <rdfs:subClassOf rdf:resource="#WineSugar"/>
    <owl:disjointWith rdf:resource="#Dry"/>
 </owl:Class>
  <owl:Class rdf:ID="RieslingSpaetlese">
   <rdfs:subClassOf rdf:resource="#LateHarvest"/>
   <rdfs:subClassOf rdf:resource="#Riesling"/>
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#hasSugar"/>
       <owl:allValuesFrom rdf:resource="#Dry"/>
     </owl:Restriction>
   </rdfs:subClassOf>
 </owl:Class>
}
```

The identifier for the new concept (*SweetLateHarvest*) was generated simply by adding the identifier of the axiom *Sweet* causing the inconsistency to the previously unsatisfiable concept *RieslingSpaetlese* .

Conclusion

In this paper, an approach to automatically resolve inconsistent ontologies was presented. In particular, a solution for overgeneralized and polysemous concepts was discussed and an experimental evaluation was sketched using standard ontologies. This approach is sustainable in the sense that it deletes as little conflicting information as possible from an unsatisfiable terminology, it remains within the syntactic framework of description logics, and it does not require any human interaction (although the human supervision of the regenezalization process is possible). We described our algorithm as well as a prototypical implementation of it. Furthermore we discussed a few examples from extending the wine-ontology with automatically learned axioms, using the *Text2Onto* toolkit.

The experiment that we have presented in this paper shows that the proposed approach is in principle relevant. But this is only a first step in evaluating the algorithm. We intend to test our approach on larger ontologies formalized in more expressive description logics. We hope that by testing on larger and more complex real-world data we get a better understanding on which types of unsatisfiability usually occur and how frequent they are. This will help to find more application-oriented solutions to the debugging problem. From a pragmatic point of view, several questions have to be answered: What is the best strategy to choose an overgeneralized concept to be rewritten? How to distinguish between overgeneralized and polysemous concepts more precise? We hope to find answers to these questions with the help of further experiments.

We think that the approach can successfully be applied to information models as well, since some types of logical inconsistencies in information modeling represent substantially terminological problems (cf. examples in [L81, BKSL01, SSJM04]). In particular, with respect to newer developments of information modeling in business applications including reasoners such techniques will play a more important role. But it is still a matter of investigation which types of inconsistencies in information modeling can be caught purely logically and whether they can be completely reduced to terminological problems.

Acknowledgments. This line of research was partially supported by the grant MO 386/3-4 of the German Research Foundation (DFG).

References

[AFFS06] Alonso, F.; Fernández, R.; Frutor, S.; Soriano, J.: Semantic Modeling of Management Information: Enabling Automatic Reasoning on DMTF-CIM. *Transactions on Engineering, Computing and Technology*, 11:24-30, 2006.

[BB04] Baroni, M.; Bernardini, S.: BootCaT: Bootstrapping Corpora and Terms from the Web. *Proceedings of LREC 2004,* 2004.

[BCMNP03] Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; Patel-Schneider, P. (eds.): *Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2003.

[BLMSW05] Baader, F.; Lutz, C.; Milicic, M.; Sattler, U.; Wolter, F.: Integrating Description Logics and Action Formalisms: First Results. *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05),* 2005.

[BKSL01] Baclawski, K.; Kokar, M.; Smith, J.; Letkowski, J.: Consistency Checking of RM-ODP Specifications. *ICEIS 2001, International Conference on Enterprise Information Systems*, Setúbal, Portugal, 2001.

[BB04] Baroni, M.; Bernardini, S.: BootCat: Bootstrapping corpora and terms from the web. *Proc. of Lexical Resources and Evaluation Conference*, 2004.

[CV05] Cimiano, P.; Völker, J. Text2Onto - a framework for ontology learning and data-driven change discovery. Proc.of *the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'2005)*, 2005.

[CBH93] Cohen, W.;Borgida, A.; Hirsh, H.: Computing least common subsumers in description logics. P. Rosenbloom and P. Szolovits, editors, *Proc. of the 10th Nat.Conf. on Artificial Intelegence (AAAI-92)*, 754–761, Menlo Park, California, 1993.

[DVBAPD04] De Vergara, L; Villagra, J.E.; Berrocal, V.A.; Asensio, J.; Pignaton, J.I.; Dpto, R.: Semantic management: application of ontologies for the integration of management

information models. *IFIP/IEEE Eighth International Symposium on Integrated Network Management*, pp. 131-134, 2003.

[E05] Egyed, A.: Instant Consistency Checking for the UML. *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, Shanghai, China, 2005.

[FFIPS04] Fanizzi, N.; Derilli, S.; Iannone, L.; Palmisano, I.; Semeraro, G.: Downward refinement in the ALN description logic. *Proceedings of the 4th International Conference on Hybrid Intelligent Systems (HIS'04)*, pp. 68-73, 2005.

[GDD05] Gašević, D.; Djurić, D.; Devedžić, V.: Bridging MDA and OWL Ontologies. *Journal of Web Engineering* 4(2): pp. 118-143.

[GLW06] Ghilardi, S.; Lutz, C.; Wolter, F.: Did I damage my ontology: A Case of Conservative Extensions of Description Logics. *Proceedings of the Tenth International Conference of Principles of Knowledge Representation and Reasoning 2006 (KR'06)*, AAAI Press, pp. 187-197, 2006.

[G98] Guarino, N.: Formal Ontology and Information Systems. In: *Proc. FOIS '98*, Trento, Amsterdam IOS Press, pp. 3-15,1998.

[HS05] Haase, P.; Stojanovic, L.: Consistent evolution of OWL ontologies. *Proc. of the 2nd European Semantic Web Conference*, Heraklion, Greece, 2005.

[HHHSS05] Haase, P.; van Harmelen, F.;Huang, Z.; Stuckenschmidt, H; Sure, Y.: A framework for handling inconsistency in changing ontologies. *Proc.of the Fourth International Semantic Web Conference*, LNCS. Springer, 2005.

[HV02] Heymans, S.; Vermeir, D.: A defeasible ontology language. *Confederated International Conferences: CoopIS, DOA and ODBASE 2002*, edited by R. Meersman, Z. Tari, pp. 1033-1046, 2002.

[KP05] Katz, Y.; Parsia, B.: Experiences and directions. Available online: *http://www.mindswap.org/2005/OWLWorkshop/sub7.pdf*.

[K06] Kalyanpur, A.: Debugging and Repair of OWL Ontologies. *Ph.D. Dissertation, University of Maryland College Park, 2006*.

[LPSV06] Lam, J.; Pan, J.; Sleeman, D; Vasconcelos, W.: A Fine-Grained Approach to Resolving Unsatisfiable Ontologies. *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI-2006)*, 428 - 434. 2006.

[L81] Lundberg, B.: On Consistency of Information Models. *BIT* 21(1): pp. 37-45, 1981.

[M01] Martin-Flatin, J. P.: Toward Universal Information Models in Enterprise Management. *Proceedings VLDB 2001 Workshop on Databases in Telecommunication (DBTel 2001)*, Lecture Notes in Computer Science, Springer, pp. 167-178, 2001.

[MSS04] Motik, B; Sattler, U.; Studer, R.: Query Answering for OWL-DL with Rules. *Proceedings of ISWC 2004*, LNCS 3298, Springer, pp. 549-563, 2005.

[M98] Mylopoulos, J.: Information Modeling in the Time of the Revolution. *Information Systems*, Vol. 23, No. 3-4, 1998.

[OK06] Ovchinnikova, E.; Kühnberger, K.-U.: Adaptive ALE-TBox for Extending Terminological Knowledge. In A. Sattar, B. H. Kang (eds.): *Proceedings of the 19th ACS Australian Joint Conference on Artificial Intelligence*, LNAI 4304, Springer, pp. 1111-1115, 2006.

[OK07] Ovchinnikova, E.; Kühnberger, K.-U.: Debugging Overgeneralized Concepts in ALCN Terminologies. In preparation.

[OWL04] Web Ontology Language (2004), Overview. W3C Recommendation 10 February 2004. *http://www.w3.org/TR/owl-features/*.

[QAWBS04] Quirolgico, S.; Assis, P.; Westerinen, A.; Baskey, M.; Stokes, E.: Toward a Formal Common Information Model. In C. Bussler et al. (eds.): *Web Information Systems – WISE 2004 Workshops*, LNCS 3307, Springer, pp. 11-21, 2004.

[SC03] Schlobach, S.; Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. *Proceedings of IJCAI'03*. Morgan Kaufmann, 2003.

[SSJM04] Simmonds, J.; Van Der Straeten, R.; Jonckers, V.; Mens, T.: Maintaining Consistency between UML Models Using Description Logic. *RSTI série L'Objet, Langages et Modèles à Objets, LMO'04*, pp. 231-244, Volume 10, 2004.

[SM01] Stumme, G; Maedche, A.: FCA-Merge: Bottom-Up Merging of Ontologies. In B. Nebel (eds.): *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence* (IJCAI'01), pp. 225-234, 2001.

[W03] Web-Based Enterprise Management (WBEM): *Technical report, Distributed Management Task Force*, 2003.

[WHRDS05] Wang, H.; Horridge, M.; Rector, A.; Drummond, N.; Seidenberg, J.: Debugging OWL-DL Ontologies: A Heuristic Approach. *Proceedings of the 4th International Semantic Web Conference 2005,* LNCS 3729, pp. 745–757, 2005.

About the Authors

Ekaterina Ovchinnikova studied Mathematical, Applied, and Computational Linguistics at the Universities of Saint-Petersburg (Russia) and Tübingen (Germany). She is currently a PhD student at the Institute of Cognitive Science, University of Osnabrück. Her research interests include formal ontologies, natural language semantics, and information retrieval from texts.

Tonio Wandmacher studied Computational Linguistics, Psychology and Computer Science (M.A.) in Tübingen and Hong Kong. After his graduation in 2004 he enrolled in a French-German PhD Programme and spent the first part at the Université de Tours until 2006. He now works as a research associate in the DFG-funded project "Adaptive Ontologies on Extreme Markup Structures" at the Institute of Cognitive Science, University of Osnabrück, where he also pursues his PhD project.

Kai-Uwe Kühnberger studied Philosophy, Linguistics, and Mathematics at the Universities of Tübingen, Stuttgart, and Bloomington, Indiana. He received his PhD in Computational Linguistics from the University of Tübingen in the year 2002. He is currently assistant professor at the Institute of Cognitive Science at the University of Osnabrück. The main focus of his research covers non-classical forms of reasoning, ontology design, neuro-symbolic integration, natural language processing, and machine learning.