RESEARCH PAPERS OF THE WROCŁAW UNIVERSITY OF ECONOMICS
No. 25 2008
Knowledge Acquisition and Management

Peter Plessers*, Krzysztof Węcel**

*Vrije Universiteit Brussel, Belgie
Peter.Plessers@vub.ac.be
**Poznań University of Economics, Poznań, Poland
K.Wecel@kie.ae.poznan.pl;

ONTOLOGY CHANGE REPREZENTATION AND MAINTENANCE FROM INFORMATION EXTRACTION

Abstract: As change is intrinsic to the surrounding world, the ontology modeling the domain of discourse should be in constant evolution. One of the sources for timely discovering of changes are documents published by specialized companies. As news documents contain mostly events, there are many approaches to extract information about instances and events in temporal context. However, there is a problem of detecting and removing inconsistencies in ontology built from text. We propose an approach based on time where we apply update patterns that change ontology according to the extracted events preserving their order, thus more precisely showing the ontology evolution. The reasoner is responsible for checking if the ontology is still consistent. Finally, conflict resolution rules are applied when necessary to restore the consistency.

Key words: ontology change, information extraction, OWL DL.

1. Introduction

Evolution is an intrinsic part of the Semantic Web [Studer et al. 1998]. Alterations in a particular domain, changes to user requirements or corrections of design flaws, they all may induce changes to the corresponding ontologies. Moreover, changes to a single ontology may have implications on many depending artifacts. In general, several problems are associated with ontology evolution. These problems include aspects such as ontology change discovery, consistency maintenance, backward compatibility, ontology manipulation, understanding of ontology evolution, change propagation, etc.

Currently, changes to ontologies are in most cases driven by human ontology engineers and domain experts. They evaluate relevant modifications to the underlying domain of an ontology, identify possible design flaws that call for a correction,

and investigate whether changing user or business requirements necessitate some ontology changes. Based on their insight, they decide which concepts of an ontology should be added or removed, and how existing concepts should be modified. The main problem ontology engineers and domain experts are facing in this aspect is that the identification of necessary changes is an extremely challenging task. The most important reasons that make this task such a difficult process are the following:

- Identifying necessary changes requires a thorough understanding of the underlying domain of an ontology. While such an understanding can be expected from a domain expert, this does not necessarily hold for an ontology engineer managing the ontology. When the task is performed by an ontology engineer lacking this deep domain understanding, the task probably turns out to be not only more time consuming, but most likely also more error-prone.
- It is very likely that ontology engineers overlook some of the relevant changes. This is due to the fact that the task is a human-only task and because ontologies can grow quite large and complex.
- When an ontology engineer decides to modify an ontology, it is difficult for him to determine whether the changed ontology correctly reflects the changes underlying domain and meets the changed user and business requirements, and which 'objectives' were possibly not met.

In this paper, we present our ongoing work on an *ontology change discovery* approach that aims to solve some of the aforementioned difficulties. Our approach is targeted to the OWL DL ontology language. The aim of our approach is to amend the current version of an existing ontology to the current reality of the underlying domain. Whenever the ontology no longer matches the underlying reality, the approach suggests possible changes to the ontology engineer to bring the ontology back in sync with reality. As computer systems are not visionary neither can they witness reality, the approach relies on a stream of Web documents as a representation of reality. These documents contain descriptions of topical events in various aspects of the underlying domain of a certain ontology. The documents are expressed in natural language and may come from different sources (e.g. news items, business reports, stock exchange documents). Information extraction techniques are used to extract instantiations of existing concepts and properties defined in a given ontology. Subsequently, the instantiations are used to verify whether they form a valid model of the current version of the ontology. If this is not the case, possible changes are suggested to adapt the ontology in order to conform to the extracted instantiations.

The structure of the paper is as follows. Section 2 discusses related work. Section 3 presents the overall architecture of our approach. Section 4 focuses on the detailed functioning of our approach, presenting the information extraction techniques, update patterns and conflict resolution rules used. Section 5 discusses the implementation of our approach, while Section 6 ends this paper with some conclusions.

2. Related work

In order to execute structured queries on text we need to employ information extraction techniques. Although the problem of extracting information is well known and many approaches have already been proposed, there is no satisfactory application for ontology evolution. There are still many challenges that hinder precise extraction of meaning. On the one hand, knowledge in documents is imprecise; on the other, ontology learning algorithms still generate uncertain knowledge. Before it is possible to analyze ontology evolution, one has to learn the initial ontology. There are many approaches to ontology learning [Maedche 2002] and most of them employ information extraction techniques; some rely purely on IR techniques over bag-of-words representation.

There are several similar projects on the topic of ontology evolution. Many of them focus on ontology learning and not necessarily on analysis of ontology evolution itself. They utilize various techniques in order to automate ontology population, leaving the consistency problem almost untouched.

One of the related systems is Text2Onto [Cimiano, Völker 2005], which aims at ontology learning. Ontology learning tasks were divided into concept extraction, instance extraction, similarity extraction, concept classification, and instance classification. The specific features of this system are incremental ontology learning, storing ontology in a proprietary format, and storing evidence's references to the source text.

Another project is ONTOTEXT [Magnini et al. 2005]. Most important features of this complex system are repository of facts, traceability, temporal binding of information, confidence of extracted information, verification of consistency.

There are also some works that focus purely on ontology evolution, without pointing what is the source of evolution. For example, Haase et al. [2005] identified several approaches to handle ontology change to avoid ontology inconsistency issue:

– maintaining consistency of initially consistent ontology by applying appropriate changes,
– repairing an inconsistent ontology,
– reasoning with inconsistent ontology (usually on consistent sub-ontology),
– finding the right version of an ontology that is consistent.

Another example is OntoAnalyzer [Rogozan, Paquette 2005] which tracks changes and formalizes them using specific language for representing ontology changes. It can also identify changes *a posteriori*. Additionally, it tries to keep annotation of the resources up to date with regard to ontological changes.

In this paper we present information extraction architecture that is suited for ontology change approach proposed by one of the authors [Plessers 2006].

3. Scenario

This section introduces a short example scenario that is used throughout the paper to illustrate the different aspects of our approach. We have developed a small OWL ontology in the domain of economics, describing concepts such as companies (including different types of companies), shares, ownership, acquisitions, merges, etc. Furthermore, it also defines concepts to capture the management of a company such as board of directors, chairman, CEO, etc. We deliberately included a number of design flaws in the initial version of the ontology in order to observe the robustness of our approach.

As the stream of documents, we have opted for two different sources. The first source delivers general business news items (including news facts about takeover purchases, changing of directors). The second source consists of stock exchange reports mainly providing information about stock market evolutions.

Based on this stream of documents, our approach allows to reveal some of the design flaws in the initial ontology version and to provide a number of solutions to adapt the ontology to the reality as described by the text documents. The working of our approach is described in the next sections.

4. Architecture

From a logical point of view, OWL ontology comprises two components: a TBox and an ABox. The TBox introduces the terminology, i.e., the *vocabulary* of the application domain, while the ABox contains *assertions* about individuals in terms of this vocabulary. The approach verifies whether a changed ABox, where the changes of the ABox are based on extracted information from a stream of text documents, is still a valid model of the TBox.

Figure 1 shows an overview of the architecture of our approach. The rounded rectangles indicate the processing steps involved, while the non-rounded rectangles indicate the different input and output forms of these processes. The role of each of the components is as follows:

Document stream: represents unstructured information, which is mostly human readable. Text originates from documents that are filtered from pre-defined Internet sources in order to restrict to selected domain, e.g. business news. Documents bring new information that has to be structured.

Information Extraction: is a process that allows to prepare information for further machine processing. It focuses on finding named-entities and relations between them. Structured representation of the documents is stored in an internal format in a database. Current trends in research point at ontologies as a way to represent information in a structured way, thus facilitating more precise querying and question answering. For information extraction we use SProUT [Piskorski 2005].

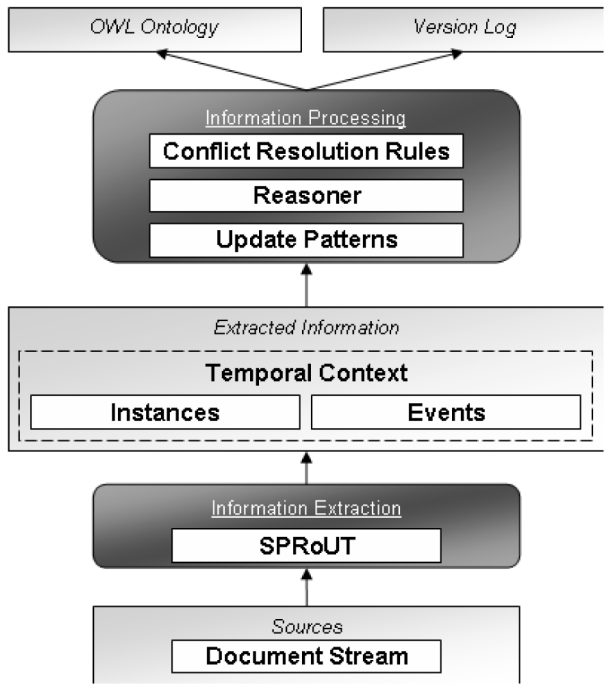

Fig. 1. Architecture

Source: own elaboration.

Extracted Information: An important issue for us to track evolution of ontology was to extract information with temporal context. Therefore, not only instances are extracted but also the time when they were certainly valid; for events the time when event occurred is necessary. In order to form an event, the relations between instances need to be discovered. New relations may pose a need to change the ontology.

Information Processing: The process creates an updated version of the ABox of the ontology based on the extracted information. The updates to the ABox are generated from a set of *update patterns*. These update patterns are used to determine whether extracted information leads to new assertions, modifications or deletions of existing assertion in the ABox. The update patterns are domain dependent. Subsequently, a *DL reasoner* is used to verify whether the updated ABox is still consistent w.r.t. the TBox of the ontology. If the ABox is no longer consistent, the approach suggests a number of possible changes based on a set of *conflict resolution rules*. The update patterns and conflict resolution rules are discussed in more detail in Sections 5.2 and 5.3.

OWL Ontology: As a result of the information processing step, a new version of the ontology may be produced, which reflects the changes in the underlying domain.

Version Log: The version log keeps track of the history of the ontology changes. For each class, property and individual that is created, it stores the versions of these concepts at the different moments in time. The version log provides ontology engineers an overview of the evolution of their ontology and allows them to return to previous versions when they disagree with changes suggested by the approach. For more details about the version log, we refer the interested reader to [Plessers 2006].

5. From IE to change discovery

This section discusses in more detail the main aspects of our approach. Section 5.1 describes the information extraction step, Section 5.2 – the update patterns, and finally Section 5.3 – the conflict resolution rules.

5.1. Information extraction

Information extraction process consists of many sub-processes which aim at structuring of text on various levels. By applying information extraction techniques, we are able to extract named entities (NEs) and structure incoming documents into contexts (temporal and spatial).

For the extraction task we use SProUT [Drozdzynski et al. 2004], a novel shallow NLP platform consisting of a pool of linguistic processing resources (tokenizer, gazetteer checker, morphology, sentence boundary detection, and partial coreference resolver) and a grammar interpreter, where the grammar formalism is a blend of efficient finite-state devices and expressive unification-based paradigm. In the previous work we have adapted and extended the existing resources for Polish, including type hierarchy, gazetteer, and named-entity grammars [Abramowicz et al. 2006].

We focus on extracting events that occur in a given location and in a given time (contexts are used). Unlike the relations, events may directly pose a need to change the ontology. For example, compare two statements: “Google owns YouTube” and “Google bought YouTube”. The first one is a relation, the latter – an event. In the case of relations the temporal information is rarely available and one has to compare information found in the document with the knowledge base to decide if the change is necessary. Events are easier to process and may be directly represented as subject-predicate-object annotated with temporal context.

The information extraction step is only for structuring of documents; all extractable events extracted from documents are stored in a database. Further analysis of what was extracted is carried out in next steps, where appropriate update patterns may be applied.

5.2. Update patterns

As explained in the previous section, the extracted information consists of a set of triples indicating the type of the individuals (e.g., <Google, type, Company>) and events that the individuals participate in (e.g., <Google, buys, YouTube>). The task of the update patterns is to transform the extracted information into changes to the ABox of the ontology.

The update patterns are simple rules consisting of an antecedent (i.e., the condition that must hold) and a consequent (i.e., the changes to be made to the ABox). An example update pattern is given below:

```
type(?a, Company) ^ type(?b, Company) ^ buys(?a, ?b)
→ addPropertyValue(?a, owns, ?b)
```

The example rule states that if it is extracted from the text documents that a company ?a buys a company ?b (where 'buys' is the event found), the property value stating that ?a owns ?b should be added to the ABox. The predicates used in the consequent of the rule are built-in change operators of our approach. In the case that the property value that is to be added is already part of the ABox, nothing is added. In a similar fashion, we can define an update patterns that reflects the selling of a company:

```
type(?a, Company) ^ type(?b, Company) ^ sells(?a, ?b)
→ deletePropertyValue(?a, owns, ?b)
```

In this case, the property value stating that ?a owns ?b is deleted from the ABox whenever it is extracted from the text documents that company ?a has sold company ?b. In the case that the property value that is to be deleted is not part of the ABox, nothing is deleted. Important to note is that the antecedent of the update patterns do not solely depend on the type of event found, but also take the types of the individuals involved in the event into account. Consider as an example the following two update patterns. Although the antecedent of both patterns includes the same event, the consequent differs because of the different types of individuals involved in the event. The first pattern declares that whenever it is extracted from text that company ?a releases a product ?b, it means that the company *offers* that product. The second pattern however declares that whenever it is extracted from text that a company ?a releases a patent ?b, it means that the company has donated that particular patent (e.g., to the open source community) and consequently no longer *owns* that patent.

```
type(?a, Company) ^ type(?b, Product) ^ release(?a,
?b) → addPropertyValue(?a, offers, ?b)
```

```
type(?a, Company) ^ type(?b, Patent) ^ release(?a, ?b)
→ deletePropertyValue(?a, owns, ?b)
```

5.3. Reasoner

In order to verify whether the updated ABox is still consistent w.r.t. the TBox of the ontology, one may use a DL reasoner. While such reasoners allow detecting inconsistencies, determining *why* the ontology is inconsistent and *how* to resolve these inconsistencies is often not supported. Nevertheless, (1) pinpointing the axioms that lead to an inconsistent ontology, (2) determining the reasons for the inconsistencies, and (3) using these reasons to offer the ontology engineer suggestions how to resolve these inconsistencies, are necessary if we want to be able to advise ontology engineers on how to adapt their ontologies to reality.

The basic principle of the tableau algorithm to check the satisfiability of a concept C is to gradually build a model I of C i.e., an interpretation I in which CI is not empty. The algorithm tries to build a tableau for the concept C by decomposing C using tableau expansion rules. These expansion rules correspond to constructors in the description logic. E.g., $C * D$ is decomposed into C and D, referring to the fact that if $a \in (C * D)I$ then $a \in CI$ and $a \in DI$. The tableau algorithm ends when either no more rules are applicable or when a clash occurs. A clash is an obvious contradiction and exists in two forms: $C(a)$ β ¬ $C(a)$ and $(\leq n\ S)$ β $(\geq m\ S)$ where $m > n$. A concept C is considered to be satisfiable when no more rules can be applied and no clashes occur. The tableau algorithm can be straightforwardly extended to support consistency checking of ABoxes as well.

In order to reveal the exact cause of a detected inconsistency, we make use of the algorithm developed by the authors of [Plessers, De Troyer 2006]. They have extended the tableau algorithm so that it becomes possible to reveal which axioms of an ontology are responsible for a detected inconsistency. It is guaranteed that deleting one of the responsible axioms (or modifying the axioms in the correct manner – see Section 5.4) resolves the detected inconsistency. The proposed extension introduces the notion of an *annotated tableau*. An annotated tableau is very similar to a regular tableau. The difference is that each label associated with a node of the tableau is annotated with the set of labels that were used by the tableau algorithm to add the particular label to the node. Consequently, the annotations that belong to the labels involved in a clash contain the path of labels that lead to the addition of these labels. Using this path of labels, it becomes straightforward to extract a set of axioms responsible for a certain clash.

We illustrate the algorithm by means of a small example. Assume the following TBox: {*Company* β *Organization* * ∀*sells*.*Product*, *Product* β ¬ *Company*} stating that a company is an organization that sells products and that a product is not a Company. It is clear that the TBox does not contain any unsatisfiable classes. Assume the following ABox is extracted from the text documents (i.e., the result of applying the update patterns): {*Company*(*a*), *Company*(*b*), *sells*(*a*, *b*)} stating that *a* and *b* are companies and *a* sells company *b*. This extracted ABox is inconsistent w.r.t. to the TBox as the individual *b* must be both an instance of *Company* and ¬ *Company*,

leading to an obvious clash. The annotated tableau when checking the consistency of the ABox w.r.t. the TBox is shown at the top in Fig. 2.

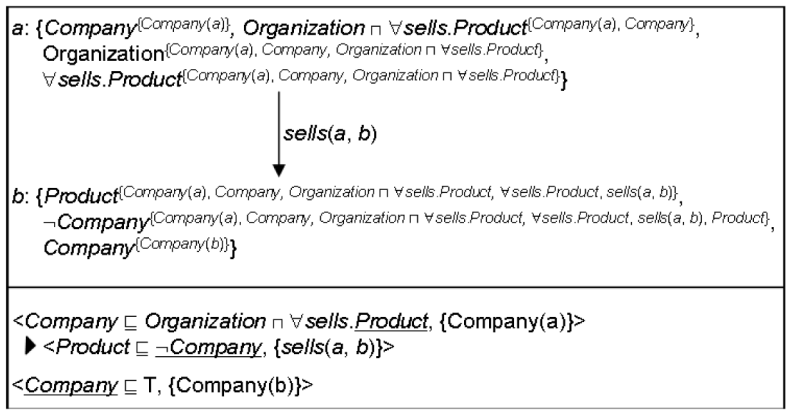

Fig. 2. An example of an annotated tableau

Source: own elaboration.

When taking the annotations of the labels involved in the clash (i.e., *Company* and ¬ *Company*), the set of axioms forming the cause of the clash can be straightforwardly retrieved. The algorithm creates a hierarchy of the concept axioms in each annotation. Role axioms and individual axioms are set as attributes of these concept axioms in the hierarchy. The hierarchies of our example are shown at the bottom of Fig. 2. The exact algorithm can be found in [Plessers 2006]. Note that the algorithm marks concepts of the concept axioms directly involved in the clash (e.g., ¬ *Company*) and concepts indirectly leading to a clash (e.g., *Product*). Marked concepts are underlined in Fig. 2. These marked concepts indicate parts of the axiom that are responsible for the detected inconsistency. For example the fact that *Company* is a subclass of *Organization* does not contribute to the inconsistency as *Organization* is not marked. The set of axioms causing an inconsistency is the set of all axioms (including concept, role and individual axioms) of both hierarchies and looks as follows: {*Company*(*a*), *Company* β *Organization* ∗ ∀ *sells*.*Product*, *sells*(*a*, *b*), *Product* β ¬ *Company*, *Company*(*b*), Company β Ø}. It can be easily seen that removing one of the axioms will resolve this particular detected inconsistency as this would break the path leading to one of the labels in the tableau involved in the clash.

5.4. Conflict resolution rules

The reason for an inconsistent ontology is that the overall set of axioms of the ontology is too restrictive in the sense that axioms are contradicting each other. A straightforward solution to overcome an inconsistency would be to simply remove one of the axioms selected by our algorithm discussed in the previous section.

However, simply removing axioms does not (in general) meet the goals of our approach as we aim to adapt the ontology to reality. Therefore, we propose a solution of weakening the restrictions imposed by the axioms in order to resolve inconsistencies. In this section, we present a set of rules that ontology engineers can use to weaken the set of axioms in order to overcome the detected ontology inconsistency.

Before we define the different rules, we first introduce the notion of a proper subclass and sub-property relationships. Assume **S** to be the set of axioms responsible for a particular inconsistency. We call Hc the class hierarchy of all classes present in the set **S** so that if $(C, D) \in$ Hc then $C \subseteq D \wedge \neg\exists Z.(C \subseteq Z \wedge Z \subseteq D \wedge Z \neq C \wedge Z \neq D)$. The property hierarchy Hp of all properties present in the set **S** is defined analogous. Note that these hierarchies do not include classes or properties not included in **S**. Furthermore, we define ψt as the top of a hierarchy H for a concept ψ, notation *top*(ψt, ψ, H), iff $\psi t \subseteq \psi \wedge \neg \exists \omega \mathbf{S}: \omega \subseteq \psi t$. Analogous, we define ψl as the leaf of a hierarchy H for a concept ψ, notation *leaf*(ψl, ψ, H), iff $\psi \subseteq \psi l \wedge \neg\exists \omega \in \mathbf{S}: \psi l \subseteq \omega$.

In the remainder of this section, we present a collection of rules that guides the ontology engineer towards a solution to overcome the inconsistency. A rule either calls another rule or requests a change to an axiom. Note that it remains the responsibility of the ontology engineer to decide which axiom of a set **S** she desires to change. Also, in the case where more than one rule is applicable, it is the choice of the ontology engineer to select the rule she finds appropriate. First, we define a set of rules that handle the different types of axioms. Next, we define the necessary rules to weaken (generalize) or strengthen (specialize) the different types of concepts.

- **Concept Inclusion Axiom:** A concept inclusion axiom $C \beta D$ can be weakened by specializing C or generalizing D:
  (5.1) *weaken*($C \beta D$) → *strengthen*(C) ∨ *weaken*(D)
- **Concept Assertion:** A concept assertion $C(a)$ can be weakened by replacing C with a superclass of C:
  (5.2) *weaken*($C(a)$) → *changeInstanceOf*(a,C,D)
  where $\exists Cl.(Cl \subseteq D \wedge Cl \neq D \wedge leaf(Cl,C,$ Hc$) \wedge D \subseteq Cl)$
- **Concept Definition Axiom**: A concept definition axiom $C \equiv D$ can be weakened either by strengthening C or weakening D, or by weakening C or strengthening D (depending on the direction in which the axiom was used in the reasoning process leading to the found clash). The first rule listed below corresponds to the direction $C \beta D$, the second rule corresponds to the direction $D \beta C$:
  (5.3) *weaken*($C \equiv D$) → *strengthen*(C) ∨ *weaken*(D)
  (5.4) *weaken*($C \equiv D$) → *weaken*(C) ∨ *strengthen*(D)
- **Transitive Axiom:** A transitive axiom *Trans*(R) can only be weakened by removing the transitivity property of R: (5.5) *weaken*(*Trans*(R)) → *deleteTransitive*(R)
- **Role Assertion:** A role assertion $R(a, b)$ can be weakened by replacing R with a superproperty of R: (5.6) *weaken*($R(a, b)$) → *changePropertyOfPropertyValue* (a, b,R, S)
  where $\exists Rl.(Rl \subseteq S \wedge Rl \neq S \wedge leaf(Rl,R,$ Hp$) \wedge S \subseteq Rl)$

- **Individual Equality & Inequality:** An individual equality $a = b$ and individual inequality $a \neq b$ can only be weakened by removing the axiom:
  (5.7) *weaken*($a = b$) → *deleteSameAs*(a, b)
  (5.8) *weaken*($a \neq b$) → *deleteDifferentFrom*(a, b)

The second part of rules deal with the weakening (generalizing) and strengthening (specializing) of concepts. A concept can be always weakened by removing the concept. We therefore will not mention this option explicitly in the rules below. When the rules to weaken a concept are similar to the rules to strengthen a concept, we omit these last rules. The rules for weakening and strengthening concepts are defined as follows:

- **Conjunction:** A conjunction $C * D$ can be weakened (strengthened) by weakening (strengthening) either C or D. The rules for weakening are given below; the rules for strengthening are analogous:
  (5.9) if Marked(C): *weaken*($C * D$) → *weaken*(C)
  (5.10) if Marked(D): *weaken*($C * D$) → *weaken*(D)
  (5.11) if Marked(C) ∧ Marked(D): *weaken*($C * D$) → *weaken*(C) *weaken*(D)
- **Existential Quantifier:** An existential quantification $\exists R.C$ can be weakened and strengthened in two manners as it represents both a cardinality restriction ("at least one") and a value restriction. To weaken $\exists R.C$, we either remove $\exists R.C$ if it concerns a cardinality restriction violation, or we weaken C if it concerns a value restriction violation. To strengthen $\exists R.C$, we either add a minimum cardinality restriction if it concerns a cardinality restriction violation, or we strengthen C if it concerns a value restriction violation:
  (5.12) if Marked(R): *weaken*($\exists R.C$) → *deleteSomeValuesFromRestr.*($\exists R.C$)
  (5.13) if Marked(C): *weaken*($\exists R.C$) → *weaken*(C)
  (5.14) if Marked(R): *strengthen*($\exists R.C$) → *addMinCardinalityRestr.*(R, 2)
  (5.15) if Marked(C): *strengthen*($\exists R.C$) → *strengthen*(C)
- **Universal Quantifier:** A universal quantification $\forall R.C$ can be weakened (strengthened) by weakening (strengthening) C. The rule for weakening is given below; the rule for strengthening is analogous:
  (5.16) *weaken*($\forall R.C$) → *weaken*(C)
- **Minimum Cardinality:** A minimum cardinality restriction ($\geq n\ R$) can be weakened by either lowering n or by removing the cardinality restriction altogether. To strengthen ($\geq n\ R$), we can raise n:
  (5.17) *weaken*(($\geq n\ R$)) → *changeCardinality*(($\geq n\ R$), m)
  where $m \leq \alpha$ if ($\geq n\ R$) conflicts with ($\leq \alpha\ R$)
  (5.18) *strengthen*(($\geq n\ R$)) → *changeCardinality*(($\geq n\ R$), m)
  where $m \geq \alpha$ if ($\geq n\ R$) conflicts with ($\leq \alpha\ R$)
- **Negation:** A negation $\neg C$ is weakened by strengthening C. To strengthen $\neg C$, we need to weaken C:
  (5.19) *weaken*($\neg C$) → *strengthen*(C)
  (5.20) *strengthen*($\neg C$) → *weaken*(C)

Atomic Concept: An atomic concept A is weakened either by removing the concept or by replacing it with a superclass of A. To strengthen an atom concept A, we replace it with a subclass of A:

(5.21) *weaken*(A) → *changeClass*(A, C)

where $\exists Cl.(Cl \subseteq C \wedge Cl \neq C \wedge leaf(Cl,A, \mathrm{Hc}) \wedge C \subseteq Cl)$

(5.22) *strengthen*(A) → *changeClass*(A, C)

where $\exists Ct.(C \subseteq Ct \wedge Ct \neq C \wedge top(Ct,A, \mathrm{Hc}) \wedge Ct \subseteq C)$

Using this set of rules to our example, we can adapt the ontology in several ways by changing one of the axioms. Which axiom to change is chosen by the ontology engineer. For example, she could decide that the assertion stating that *a* is a company does not reflect the correct real world situation. Using rule 5.2, she may decide to weaken the assertion *Company*(*a*) by changing it to *Organization*(*a*). Another example would be that the ontology engineer instead decides that the axiom *Company* β *Organization* ∗ ∀ *sells*.*Product* does not reflect reality. Using rules 5.1 and 5.16, the axiom can be weakened be replacing it with *Company* β *Organization*. In both cases, the use of these set of rules guarantee that the detected consistency will be resolved.

6. Conclusion and future work

This work in progress paper presented an approach to merging into one framework two approaches: ontology change formalism based on OWL DL and information extraction system that was able to populate the ontology but lacked a formal approach to ontology change. We therefore introduced a process that consisted of update patterns, consistency checking by a reasoner, and conflict resolution by applying appropriate weakening rules.

The main advantage of the proposed system is that need for change is extracted automatically while retaining a user the possibility to decide how ontology should evolve. For this purpose the user is presented with several conflict resolution rules from which she may choose the most suitable action. When we leave ontology learning unattended soon the ontology may become far from accurate.

The future work will focus on refining the bottom part of the framework. Information extraction rules and update patterns need to be extended to include more entity types in the ontology.

An interesting by-product of the project will be the auto-verification of the proposed framework. Information extraction rules will focus on Polish and upper part is developed by person who does not know Polish. The reasoner and conflict resolution rules are language independent but discussions on finding common representation enhanced our approach.

Acknowledgement

This research project has been supported by a Marie Curie Transfer of Knowledge Fellowship of the European Community's Sixth Framework Programme under contract number MTKD-CT-2004-509766 (enIRaF).

References

Abramowicz W., Filipowska A., Piskorski J., Węcel K., Wieloch K. (2006). Linguistic suite for Polish cadastral system. In: *5th International Conference on Language Resources and Evaluation*. Genoa, Italy, pp. 2518-2523. European Language Resources Association (ELRA).

Cimiano P., Völker J. (2005). Text2Onto. A framework for ontology learning and data-driven change discovery. In: *Proceedings of NLDB'0. Lecture Notes in Computer Science*, vol. 3513, Springer, Alicante, 2005.

Drozdzynski W., Krieger H.-U., Piskorski J., Schäfer U., Xu F. (2004). Shallow processing with unification and typed feature structures – foundations and applications. *Künstliche Intelligenz*, vol. 1, pp. 17-23.

Haase P., Harmelen F., Huang Z., Stuckenschmidt H., Sure Y. (2005). A framework for handling inconsistency in changing ontologies. In: *4th International Semantic Web Conference, ISWC 2005 Galway.* Eds. Y. Gil, E. Motta, R. Benjamins, M. Musen. Springer, LNCS, vol. 3729.

Maedche A. (2002). *Ontology Learning for the Semantic Web*. The Kluwer International Series in Engineering and Computer Science; SECS 665. Kluwer Academic Publishers, Boston.

Magnini B., Negri M., Pianta E., Romano L., Speranza M., Serafini L., Girardi C., Bartalesi V., Sprugnoli R. (2005). From text to knowledge for the semantic web: The ONTOTEXT project. In: *Semantic Web Applications and Perspectives. CEUR Workshop Proceedings.* Eds. P. Bouquet, G. Tummarello. Trento.

Piskorski J. (2005). Named-entity recognition for Polish with SProUT. In: *Intelligent Media Technology for Communicative Intelligence: Second International Workshop – IMTCI 2004, Warsaw*, Poland. Springer, LNCS, vol. 3490.

Plessers P. (2006). *An Approach to Web-based Ontology Evolution.* Doctoral Dissertation, Department of Computer Science. Vrije Universiteit Brussel, Brussels.

Plessers P., De Troyer O. (2006). Resolving inconsistencies in evolving ontologies. In: *European Semantic Web Conference*. Eds. Y. Sure, J. Domingue. Springer, LNCS, vol. 4011.

Rogozan D., Paquette G. (2005). Managing ontology changes on the semantic web. In: *Proceedings of 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pp. 430-433. IEEE Computer Society, Washington, DC.

Studer R., Benjamins R., Fensel D. (1998). Knowledge engineering: principles and methods. *Data: Knowledge Engineering*, vol. 25, no. 1-2, pp. 161-197.