Ontology Evolution within Ontology Editors

L. Stojanovic, B. Motik

FZI - Research Center for Information Technology at the University of Karlsruhe,
Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
{Ljiljana.Stojanovic, Boris.Motik}@fzi.de

Abstract. An ontology over a period of time needs to be modified to reflect changes in the real world, changes in the user's requirements, drawbacks in the initial design, to incorporate additional functionality or to allow for incremental improvement. Although changes are inevitable during the development and deployment of an ontology, most of the current ontology editors unfortunately do not provide enough support for efficient copying with changes. Since changes are the force that drives the evolution process, in this paper we discuss the requirements for the ontology editors in order to support ontology evolution.

1 Introduction

Ontologies aim at capturing domain knowledge in a generic way, and provide a commonly agreed understanding of a domain, which may be reused and shared across applications and groups. Although there are several approaches for a semi-automatic ontology development ([6], [8]), most of the existing ontologies are created manually using ontology editors.

Ontology editors are tools that enable inspecting, browsing, codifying, and modifying ontologies and support in this way the ontology development and maintenance task [11]. Existing editors vary in the complexity of the underlying knowledge model, usability, scalability, etc. Nevertheless, all of them provide enough support for the initial ontology development. However, ontology development is necessarily an iterative and a dynamic process [1]. Very seldom is an ontology perfect the first time it is made, and then continues, without change, to be as useful over time as it was when it was first deployed. The reasons for changes are inherent in the complexity of reality and in the limited ability of humans to cope with this complexity. Thus, ontologies must be able to evolve for a number of reasons, including the following:

- Ontologies often contain "design error" and sometimes do not immediately meet the requirements of its users;
- The environment in which the ontology operates can change unpredictably, thereby invalidating the assumptions that were made when the ontology was built;
- Users' requirements can change after the ontology is initially built, requiring that the existing ontology evolve to meet the new requirements.

The necessity to support change management can be derived from many real-word applications, since they typically operate in changeable environments. A typical example is MEDLINE database containing over 11 million references to articles from 4,600 worldwide journals in life sciences. It is in the irregular operation in November and December as NLM makes the transition to a new year of Medical Subject Headings[1] (MeSH), somewhere called medical ontology. Another example is UNSPSC[2] classification of products currently consisting of the hierarchy of about 16.000 product categories. Every two weeks, a change is performed, which alters between 100 and 600 concepts. This causes serious problems for companies that use it to classify their product data to allow e-commerce [4].

Therefore, the methods and tools for the ontology evolution enabling coping with the changes in a more systematic way have become an essential requirement for an ontology-based application [1]. The ontology evolution [4] is the timely adaptation of the ontology as well as the consistent propagation of these changes, because a modification in one part of the ontology may generate subtle inconsistencies in other parts of the same ontology, in the instances, depending ontologies and applications.

The ontology evolution is becoming more important nowadays. The major reason for this is the increasing number of ontologies in use and the increasing costs associated with adapting them to changing requirements. Developing ontologies and their applications is expensive, but evolving them is even more expensive. However, even though evolution over time is an essential requirement for useful ontologies [11], appropriate tools and strategies for enabling and managing evolution are still missing. This level of ontology management is necessary not only for the initial development and maintenance of ontologies, but is essential during deployment, when scalability, availability, reliability and performance are absolutely critical [1].

Since an ontology is usually developed using an ontology editor, many requirements for the ontology evolution have to be part of the ontology editors. An ontology editor must provide an interface that allows the knowledge engineer to modify the underlying ontology. The interface is based on the set of available ontology changes. Moreover, there are many features which can significantly improve the usability of an ontology editor and enhance its functionality regarding the ontology evolution. In this paper, we discuss the most critical requirements for ontology editors in order to be more robust to a changing environment.

The paper is organised as follows: In the second section, we elaborate a set of requirements for an ontology editor to be able to support ontology evolution. The evaluation of the some ontology editors in terms of these requirements is given in section 3. Section 4 contains concluding remarks.

2 Requirements for the Ontology Evolution

Ontology development is a dynamic process [7] starting with an initial rough ontology, which is later revised, refined and filled in the details [5]. Consequently, an ontology almost certainly should be evolved[3] in order:

---

[1] MeSH is a controlled vocabulary thesaurus used to index the articles [http://www.nlm.nih.gov/pubs/factsheets/medline.html]

[2] http://eccma.org/unspsc/

[3] IEEE 1219 1993

- to fix “bugs” in the initial design (corrective maintenance);
- to adapt itself to the changes in the environment (adaptative maintenance);
- to improve itself after it has become operational (perfective maintenance);
- to avoid future changes and to alleviate maintenance (preventive maintenance).

Moreover, ontology evolution has to be supported through the entire lifecycle [11]. Since ontology editors are the main tools for ontology development, the support for evolution should be a required facility in an ontology editor. In other words, the functional specification of an ontology editor has to incorporate requirements for the ontology evolution. In this paper, we have identified a set of requirements for ontology editors to allow users to be able to alter an ontology in a more efficient and convenient manner. These requirements can be divided in several groups:

- Functional requirement specifies all evolution changes that must be supported;
- User’s supervision requirement enables the user-driven process of change resolving;
- Transparency requirement deals with providing control of the evolution process through an insight into the scope of an evolution operation before the operation is applied;
- Reversibility requirement states how the effect of evolution changes can be undone;
- Auditing requirement is related to the management of the ontology change history;
- Ontology refinement requirement provides support for continual ontology improvement;
- Usability requirement allows the user to manage changes more easily by finding ontology inconsistencies and providing the explanation to solve them.

In the rest of this section, we elaborate these requirements in more details.

2.1 Functional requirement

The functional requirement specifies which functionality must be provided for the ontology development and evolution. This functionality heavily depends on the underlying ontology model. The more powerful and expressive model requires a richer set of modelling primitives. Thus, before speaking about functional requirements, the notion of an ontology itself has to be clarified. Corresponding to the variety of ontology models in use[4], there is no standard ontology model. However, an attempt to provide the standard for ontology structure[5] is on the way.

Due to differences in ontology models, we concentrate on the “common” features of ontology models, namely concepts, properties, instances, as well as concept inheritance. Each of these ontology entities can be updated by one of the meta-change transformations: add, remove, modify [3]. A full set of changes (Tab. 1) can thus be defined by the cross product of the set of entities of the ontology model, which form meta schema, and the set of meta-changes.

[4] http://www.ontoknowledge.org/oil/, http://www.daml.org/2001/03/reference.html
[5] http://www.w3.org/2001/sw/WebOnt/

Table 1. Changes in the ontology

Meta changes Meta enitities	**Add**	**Remove**	**Modify**
Concept	Add concept	Remove concept	Rename concept
Concept hierarchy	Add subConceptOf relationship	Remove subConceptOf relationship	Set subConceptOf relationship
Property	Add property	Remove property	Rename property
Property Domain	Add property domain	Remove property domain	Set property domain
Property Range	Add property range	Remove property range	Set property range
Instance	Add instance	Remove instance	Rename instance
Property Instance	Add property instance	Remove property instance	Set property instance

The existence of the "modify" change causes the set of primitives not to be minimal with respect to completeness. However, this change adds some important semantic variations to the set of changes, since a modify change is not equivalent to a removal followed by an addition [3]. The difference is that the modification of an entity (i.e. renaming a concept) maintains its identity, while removing and adding loses its identity.

The previously mentioned changes are called elementary changes, since they cannot be decomposed into simpler changes. The basic functionality of each ontology editor from the ontology evolution point of view is specified as a set of elementary ontology changes derived from the corresponding ontology model.

Elementary changes in the ontology specify fine-grained changes that can be performed in the course of ontology evolution. However, this granularity of ontology evolution changes is not always appropriate. Often, the intent of the changes may be expressed on a higher level. Composite changes [9] specify coarse-grained changes that can be performed to improve the ontology structure according to some criteria. They are more powerful, since the designer does not need to go through every step of the sequence of basic changes to achieve the desired effect.

Moreover, composite changes often have more meaningful semantics. For example, the semantics of moving the concept from one parent concept to another is clearly different from the semantics of removal and addition of a subConceptOf relation. While "move" as a composite change maintains the identifiers of a subConceptOf relation and preserves all properties and instances, the removal and subsequent addition create a new identifier for a subConceptOf relation and cause the loss of much information (e.g. at the instance level).

All valid changes to manipulate an ontology can be specified by one elementary or composite change or by a sequence of changes. Changes can be applied to an ontology in a valid state, and after all changes are performed, the ontology and dependent artefacts must transition to another valid state. It means that every change is guaranteed to maintain some constraints. We have identified the following set of system integrities that have to be maintained during resolution in order to achieve the soundness:

- Consistency – A consistent ontology is one that satisfies all invariants of the ontology model. Invariants are constraints that must hold in every quiescent state of an ontology. For example, the concept hierarchy is a direct acyclic graph;
- Validity – We distinguish between syntax and semantic validity [9] of an ontology. Syntax invalidity arises when undefined entities are used or model constraints are invalidated. Semantic invalidity arises when the meaning of an ontology entity is modified. On the other hand, a valid instance is one that conforms to the constraint specified in an ontology;
- Well-formedness - A well-formed ontology and instances are those, which syntactically conform to the language specification.

2.2 User's supervision requirement

The ontology evolution is a process of changing an ontology while maintaining its consistency. The goal of the ontology evolution is thus to evolve an ontology from one consistent stage to the next. However, there are many ways to achieve consistency after a change request. For example, when a concept from the middle of the hierarchy is being deleted, all subconcepts may either be deleted or reconnected to other concepts [9]. If subconcepts are preserved, then properties of the deleted concept may be propagated, its instances distributed, etc. Thus, for each change in the ontology, it is possible to generate different sets of additional changes, leading to different final consistent states.

Hence, a mechanism is required for users to manage changes resulting not in an arbitrary consistent state, but in a consistent state fulfilling the user's preferences. In order to enable the user to obtain the ontology most suitable to her needs, an ontology editor should allow the customisation of the ontology evolution process. One mean is to enable the user to set up one of evolution strategies that are used for resolving the changes.

An evolution strategy unambiguously defines the way how elementary and composite changes will be resolved. Typically, a particular evolution strategy is chosen by the user at the start of the ontology evolution process. Thus, an evolution strategy defining a common policy must be chosen to specify how to handle each of the following situations:

- how to handle orphaned concepts - those concepts that don't have parents any more;
- how to handle orphaned properties - those properties that don't have parents any more;
- how to propagate properties to the concept whose parent changes;
- what constitutes a valid domain of a property;
- what constitutes a valid range of a property;
- whether a domain (range) of a property can contain a concept that is at the same; time a subconcept of some other domain (range) concept;
- the allowed shape of the concept hierarchy;
- the allowed shape of the property hierarchy;
- must instances be consistent with the ontology.

For each of these situations, there is a set of possible options, e.g. in case of the first issue, orphaned subconcepts of a concept may be connected to the parent concept(s) of that concept, connected to the root concept of the hierarchy or deleted as well.

2.3 Transparency requirement

A change in one part of an ontology may have far reaching consequences on other parts of the ontology and associated instances. If an ontology is large, it may be difficult to fully comprehend the extent and meaning of each change. To improve understanding of effects of each change, the ontology evolution should provide maximum transparency into details of each change being performed. Transparency should provide a human-computer interaction for evolution by presenting change information in an orderly way, allowing easy spotting of potential problems and alleviating the understanding of the scope of the change.

Before any change is applied to the ontology, a list of all implications must be generated and reported to the user. The ontology engineer should be able to comprehend the list, and approve or cancel the change. If the changes are cancelled, the ontology should remain intact. The presentation of changes has to follow the progressive disclosure principle: related changes have to be grouped together and organised in a tree-like form. The user can initially see only the general description of changes. If she is interested in details, she can expand the tree and view complete information. She may cancel the operation before it is actually performed.

2.4 Reversibility requirement

As mentioned, the transparency requirement was introduced to help the ontology engineers comprehend the effect of a change. If properly done, this can help in reducing the number of accidental ontology changes, and can even guide the ontology refinement process. However, there are numerous circumstances where it may be desired to reverse the effects of changes. The reversibility requirement states that an ontology editor has to allow undoing changes at the user’s request. Consequently, the user can control changes and make appropriate decisions.

It is important to note that reversibility means undoing all effects of a change, which may not be the same as simply requesting an inverse change manually. For example, if a concept is removed from a concept hierarchy, its subconcepts will be modified (e.g. attached to the root). Reversing such change is not equal to recreating the deleted concept – one also needs to revert the concept hierarchy into an original state.

To support the undo-redo in a usable fashion, undoing an action must be accompanied by restoring the state of the UI to what it was before the action was performed. For example, if a concept in the concept-hierarchy tree was selected and then deleted, when the change is undone, the same concept must be selected. If the tree was scrolled in the meanwhile, the original scroll position of the tree must be restored (or at least the node must be made scrolled into view). For the navigation in an application, users often rely on the visual features of the application. When an operation is undone, it is essential to restore the previous visual state of the application as close as possible, allowing the users to quickly recognise a familiar state and proceed with their work. If the visual state of the application is not restored well, although the action is undone, the user may not realise this, and may mistakenly request another undo operation.

2.5 Auditing requirement

As business applications of ontologies proliferate, so do the needs for auditing ontology evolution. Changes to business information are often accompanied with responsibility for their effects on the business. Auditing is therefore a typical component of business systems, and must be reflected in the ontology evolution as well.

The ontology evolution auditing involves the following aspects:

- Keeping a detailed log of all performed changes allowing later reconstruction of the events that led to the current state of the ontology;
- Associating meta-information with each log change, such as textual change description, cost of change, time of change etc.;
- Tracking the identity of the change author.

The auditing requirement is also related to the reversibility requirement, since the auditing log is typically used to provide reversibility. The auditing log can also serve as a source for information mining about change trends.

2.6 Ontology refinement requirements

This requirement states that potential changes improving the ontology may be discovered semi-automatically from the ontology-based data and through the analysis of the user's behaviour. We distinguish (i) structure-driven, (ii) data-driven and (iii) usage-driven change discovery [10].

(i) The structure-driven change discovery identifies the following set of heuristics to improve an ontology based on the analysis of the structure of the ontology:

- If all subconcepts have the same property, the property may be moved to the parent concept;
- A concept with a single subconcept should be merged with its subconcept;
- If there are more than a dozen subconcepts for a concept, then an additional layer in the concept hierarchy may be necessary;
- The concept without properties is a candidate for deletion;
- If a direct parent of a concept can be achieved through a non-direct path, then the direct link should be deleted.

(ii) The data-driven change discovery states that some changes are implicit changes in the domain, reflected in its instances and can be discovered only through their analysis. We have found the following set of heuristics:

- A concept with no instances may probably be deleted;

- If no instance of a concept C uses any of the properties defined for C, but only properties inherited from the parent concept, we can make an assumption that C is not necessary;
- A concept with many instances is a candidate for being split into subconcepts and its instances distributed among newly generated concepts.

(iii) The usage–driven ontology evolution takes into account the usage of the ontology in the knowledge management system [10]. It is based on the analysis of the users' behaviour in two phases of a knowledge management cycle: in providing knowledge by analysing the quality of annotations, and in searching for knowledge by analysing the users' queries and the responses from the knowledge repository. For example, by tracking when the concept was last retrieved by a query, it may be possible to discover that some concepts are out of date and should be deleted or updated.

2.7 Usability requirement

An ontology editor addresses the issue of presenting ontologies and allowing the user to operate on ontologies in a consistent way. It also addresses how different functions are integrated into the system in a way natural to the user. An ontology editor has to have an interface that enables the user to create and maintain ontologies, one that is easily understood and allows the user to work efficiently with all the complexities inherent in an ontology editor.

However, the real usability of an ontology editor cannot be achieved only through the graphical means for the creation/modification of ontology entities which relieve the user of the necessity to perform this task manually. An ontology editor has to guide the user through the ontology development process by providing additional information, such as why the user's activity did not succeed, or what else she has to do in order to finish the current activity.

Moreover, a good ontology editor must provide capabilities for identifying inconsistencies. When such conflicts arise, an editor must assist the user in identifying the source of the problem and resolving it. Furthermore, the usability of an ontology editor can be significantly increased by incorporating validation. Validation concerns the truthfulness of an ontology with respect to its problem domain - does the ontology represent a piece of reality and the users' requirements correctly? One technique for supporting validation is generating explanation.

3. Evaluation

Ontology editors are tools that allow users to visually manipulate ontologies. The number of tools for building ontologies developed in the last years is high[6]. In this section, we evaluate three ontology editors which are most frequently used in the Semantic Web community, in terms of the requirements for the ontology evolution. Table 2 shows the result of comparison.

[6] http://www.ontoweb.org/download/deliverables/D13_v1-0.zip

Table 2. Evolution support within ontology editors. Description: “-“ means that there is no support, “<>” states that support is partial and “+” corresponds to the full support.

Editors/ Requirements	**Protege**[7]	**OntoEdit**[8]	**OilEd**[9]
Functionality			
elementary	+	+	+
composite	-	<>	-
Supervision	-	-	-
Transparency	-	<>	-
Reversibility	<>	<>	-
Auditing	<>	-	<>
Refinement	-	-	-
Usability			
user-friendly	+	+	+
verification	<>	<>	<>
validation	-	-	-

The basic functionality of each ontology editor is specified as a set of elementary ontology changes. Thus, all editors allow such modifications. Even though composite changes allow an ontology engineer to update an ontology without having to find the right sequence of elementary modifications, most of the existing ontology editors do not include composite changes. Only OntoEdit provides support for some composite changes (move and copy).

Most of the existing systems for the ontology development provide only one possibility for realising a change, and this is usually the simplest one. For example, the deletion of a concept always causes the deletion of all its subconcepts. It means that users are not able to control the way changes are performed (supervision).

Moreover, users do not obtain explanations why a particular change is necessary (transparency). In OntoEdit, the user only obtains the information about numbers of induced changes, but without providing more details.

Furthermore, there is no possibility to undo effects of changes (reversibility). Protégé and OntoEdit have Edit menu with Undo/Redo options, but they are disabled.

Regarding the auditing requirement, OilEd has the activity log. However, it records connections to the reasoner, not all ontology modifications. Protégé also has the command history option, but it is useless, since it is disabled.

As known to authors, none of the existing systems for ontology development and maintenance offer support for (semi-)automatic ontology improvement, even though it makes the ontology easier to understand and cheaper to modify.

Most of the existing ontology editors have a very similar layout. They are ergonomically correct to minimise human errors. They enable operating “quickly” enough, as this is often considered being one of the most important easy-for-use

[7] http://protege.stanford.edu/
[8] http://www.ontoprise.de/com/co_produ_tool3.htm
[9] http://oiled.man.ac.uk/

issues. Moreover, all editors can detect logical conflicts (verification), but they do not provide enough information to analyse the sources of conflicts. However, none of the existing editors provide the means to answer to the questions such as how, why, what if, etc. (validation).

4. Conclusion

In order to enable the user to obtain the ontology most suitable to his or her needs, we investigate the requirements for an ontology editor in order to customise the ontology evolution process. We identify several means to do that: to enrich the list of possible changes; to enable the user to set up one of the evolution strategies that are used for resolving the changes; to inform her about all effects of a change; to allow undoing changes; to allow inspecting the performed changes; to suggest the user to generate a change and to identify inconsistency and to provide answers to the questions such as how, why, what if, etc.

We believe that an ontology editor that fulfils these requirements will enable maintaining an ontology more easily and according to the user's preferences.

References

1. A. Das, W. Wu, D. McGuinness, *Industrial Strength Ontology Management'*, The Emerging Semantic Web*,* IOS Press, 2002.
2. D. Fensel, *Ontologies: Dynamics Networks of Meaning*, In Proceedings of the the 1st Semantic web working symposium, Stanford, CA, USA, July 30th-August 1st, 2001.
3. W. Huersch, *Maintaining Consistency and Behaviour of Object-Oriented Systems during Evolution*, PhD thesis, College of CS, Northeastern University, Boston, 1995.
4. M. Klein and D. Fensel, *Ontology versioning for the Semantic Web*, Proc. International Semantic Web Working Symposium (SWWS), USA, 2001.
5. N. F. Noy, D. McGuinness, *Ontology Development 101: A Guide to creating your first Ontology*, Stanford KSL Technical Report KSL-01-05, 2000.
6. A. Maedche and S. Staab, *Ontology Learning for the Semantic Web*, IEEE Intelligent Systems, 16(2), March/April 2001. Special Issue on Semantic Web, 2001.
7. S. Staab, H.-P. Schnurr, R. Studer and Y. Sure, *Knowledge Processes and Ontologies*, IEEE Intelligent Systems. 16(1), Special Issue on KM, 2001.
8. L. Stojanovic, N. Stojanovic and R. Volz, *Migrating data-intensive Web Sites into the Semantic Web*, In ACM Symposium on Applied Computing SAC, 2002.
9. L. Stojanovic, A. Maedche, B. Motik, N. Stojanovic, *User-driven Ontology Evolution management*, In Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management EKAW, Madrid, Spain, 2002.
10. N. Stojanovic, L. Stojanovic, *An Approach for the Evolution of Ontology-based Knowledge Management Systems*, EKAW'2002 Workshop on Knowledge Management through Corporate Semantic Webs, 2002.
11. Y. Sure, *On-To-Knowledge -- Ontology based Knowledge Management Tools and their Application*, In: German Journal Kuenstliche Intelligenz, Special Issue on Knowledge Management (1/02), 2002.