Building an Integrated Framework for Ontology Evolution Management

Asad Masood Khattak, Department of Computer Engineering, Kyung Hee University, Korea
asad.masood@oslab.khu.ac.kr

Khalid Latif, School of Electrical Engineering and Computer Science, NUST, Pakistan
khalid.latif@seecs.edu.pk

Sung Young Lee, Department of Computer Engineering, Kyung Hee University, Korea
sylee@oslab.khu.ac.kr

Young-Koo Lee, Department of Computer Engineering, Kyung Hee University, Korea
yklee@khu.ac.kr

Tahir Rasheed, Department of Computer Engineering, Kyung Hee University, Korea
tahir@oslab.khu.ac.kr

Abstract

Ontology engineering, evolution, and maintenance are collaborative processes. The crucial task is how to accommodate the new changes in the ontology while preserving its consistency. We provide here a framework for ontology evolution. As new concept(s) emerge, the proposed framework automates the process of how these new changes will be detected and then committed to the ontology. The change in focus can be a single concept, group of concepts, and/or concepts in a hierarchical structure. A log of all the implemented changes is maintained using Change History Log (CHL) with conformance to Change History Ontology (CHO) to eliminate conflicts and to support the undo and redo operations.

Keywords: Ontology, Ontology Evolution, Change Log, Emerging Concepts, Change History Ontology

1. Introduction

Ontologies are *formal description of shared conceptualization of a domain of discourse*. They evolve with the passage of time as humans develop better understanding of their perceived knowledge [17]. In other words, ontology evolution takes place when the perspective under which the domain is viewed has changed [12]. The evolution process deals with the growth of the ontology and capturing new information. More specifically, ontology evolution means *modifying or upgrading the ontology when there is a certain need for change or there comes a change in the domain knowledge*.

The process of evolution takes an ontology from one consistent state to another [6]. In doing so, the process may involve different strategies such as merging and integration [4]. The evolution process has several subtasks, which are:

1. The first step is to *capture the required change(s)* to be applied to the ontology.
2. Consequently, all the required *changes are described* using a common representation format.
3. The effects of the required changes are tested on the ontology for *consistency* and if required some *deduced changes* are also included in the change. All these deduced changes become part of the required changes.
4. The complete change request is executed by *implementing the changes* in the ontology.
5. Change verification subtask then validates the subject ontology to confirm that the requested changes have been committed to the ontology.
6. Finally, the changes are propagated to all the dependent artifacts.

The current ontology evolution techniques have several weaknesses, such as: manual specifications of new changes, manually resolving inconsistencies and/or selecting deduced changes from available alternatives, and also the absence of proper and complete undo and redo facilities. These weaknesses need to be eliminated in order to automate the ontology evolution process to the available extent. Automation of the process is also necessary because human intervention in the evolution system tends to be time consuming and error prune.

The goal of this research is to build a framework which will automatically detect the changes to be made to the ontology triggered by the change request. To ensure the consistency of the ontology, the proposed framework generates deduced changes after analyzing the change semantics based on the work presented in [16]. Finally the changes are implemented and logged using Change History Ontology to provide undo/redo functionality.

This paper is arranged as follows: Section 2 describes the existing research work in the field of ontology evolution and change management. Section 3 presents the Change History Ontology. In Section 4 we present our proposed framework of ontology evolution and implementation details and

results are discussed in Section 5. Finally we conclude our findings in Section 6 and provide an outlook of the future directions.

2. Related Work

Research on ontology evolution is being carried out by different researcher groups. In [17], the author proposed a six phase ontology evolution process which copes with the ontology changes due to business requirements and dynamic environment. It first systematically analyses the reasons for changes and makes a complete request for change while preserving consistency of the ontology and the dependent artifacts, and then implements all the changes. In [10], the author presented a framework for change management and ontology evolution for distributed ontologies. It provides way to formally describe ontology changes required to perform in evolution of ontology.

The version log concept has been given in [16], which is providing an evolution procedure while dealing with different versions of the ontology. It also is a five phase evolution procedure which are change request, change implementation, change detection, change recovery, and change propagation.

New concepts discovery process is given in [1]; it supports the ontology enrichment activity for multimedia ontology evolution. Here the main contribution is the automatic discovery of new concepts with help of ontology matching techniques presented in [3], from multimedia objects/resource with additional metadata.

Requirements that an ontology management system should provide for ontology evolution are given in [19]. The author also provided a formal model for handling semantics of change included in the ontology evolution process of OWL ontologies.

D. Oberle in [15] provides a six phase (discussed in Section 1) ontology evolution technique, for business oriented ontology management. During the change implementation, all the changes which are performed to ontology are logged for the purpose of undoing changes and provide facility of recovery.

A detail view of the methods and tools available for ontology evolution and their working is provided in [18]. In [9], the author presented an exhausted view of change management activity and also provided the different kinds of changes that can occur in ontologies, and detail of tracking different changes is available in [13] and [14].

The existing systems for ontology evolution do not consider new emerging concept(s). In [16] they are manually creating requests for change. [17],[15] needs ontology experts for conflict resolutions, and [1] after discovering a new concept needs ontology expert to insert the concept at suitable place, while in our system these all are done automatically.

3. Change History Ontology

A number of changes, ranging from concepts to properties, could affect the ontology. Most of these changes are discussed in greater length in [1]. Here we will briefly highlight some of the critical changes in concepts:

- *New Concept:* This is the most common change in any ontology. New concepts emerge and have to be accommodated in the concept hierarchy.
- *Concept with Changed Properties:* This is the case when the concept in focus is already present in the ontology but its properties and restrictions are dissimilar from those associated with existing concepts.
- *Simple vs. Aggregated Concept:* The concept in focus might be a combination of two or more existing concepts (or vice versa). The ontology framework shall preferably detect and act accordingly to accommodate the change.
- *Concept vs. Property:* Different modeling approaches are followed by ontology engineers for building ontologies. One such case is modeling the same concept either as a class in OWL or as a property of some other existing class. For example, the concept *deliverable* could be a separate class or could be modeled as property of the concept *project*. In the first case it could have been implemented as a subclass of *document* and in the second case it could take the instances of *software* as its value.

Understanding of change types is necessary to correctly handle explicit and implicit change requirements [7], and also to engineer the Change History Ontology (CHO). The baseline for CHO is the Log Ontology presented by Yaozhong David Liang [11]. We have modeled quite a few extensions to the baseline and come up with CHO as shown in Fig 1 where detail is given in [21]. Some of the extensions include:

- Capturing such provenance information as the change author, reason, timestamp.
- We introduced a class *OntologyChange*. It has further sub classes including *AtomicChange* and *ChangeSet*. The *AtomicChange* tackles all types of changes that can be applied to an ontology at its class and property levels.
- *ChangeSet* holds information about the changes whether it is an instant or composite and stretched change over a defined time interval. All instant changes are considered members of the change set for the stretched changes. *ChangeSet* also helps in properly maintaining the sequences of the changes. With the help of *ChangeSet*, all the changes of some defined time interval is organized and managed together, which in future help us to roll-back and roll-forward the changes and get the previous state of ontology.

Fig 1. Snapshot of CHO

4. Ontology Evolution Framework

Here we present the proposed framework for ontology evolution management. The aim is to develop interfaces for the baseline modules and to integrate functionality of existing components dealing with change detection and description, inconsistencies detection, or change implementation and verification. Fig 2 depicts an overview and interconnection of the components. In a holistic manner, these modules ensure that the ontology has evolved to a consistent new state incorporating all the required changes. The working details of these modules are given in the subsequent sections.

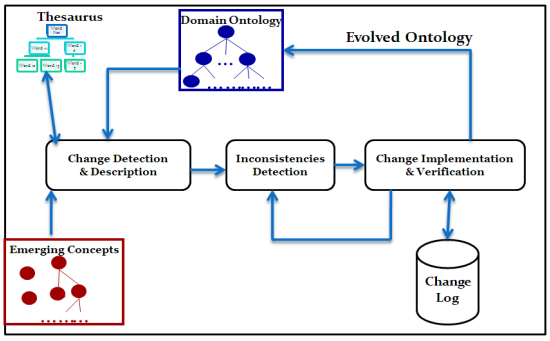

Fig 2. Overview of framework architecture

Input for the evolution process is the domain ontology. An ontology, as described earlier, is not a static entity and will change as new concepts emerge in the domain or other reasons. The change may take place because of a single concept, group of concepts, or even concepts in a hierarchical structure. Source of the change is not restricted as new concepts can spring from, for instance, the change request by the domain expert or ontology engineer.

Change Detection & Description

The first step in the process is to detect changes. For example, it has to be detected whether the new additions are already present in the target ontology. Additionally, schema and individual level differences can be detected effectively as reported in [20]. In case the concept in focus is totally new and there is no additional information, the H-Match algorithm [3] is used. Its Semantic Affinity measure provides the contextual matching facility through a set of four models namely: surface, shallow, deep, and intensive. It takes the new concepts for addition and the target ontology as input and returns the best matching concept in the ontology in order to identify taxonomic position for the concept [2]. In either case, automatic detection through H-Match algorithm or through ontology engineer's input, the change detection module make the identification of the change target certain with respect to the ontology.

After this every identified change is represented in a consistent format, where these changes may be elementary (atomic), or composite. Changes are first assembled in a sequence, and then this sequence is followed for the change implementation. We are following the steps for atomic changes and considering all the composite changes as an ordered sequence of atomic changes. Finally, the changes are represented using CHO. The same representation is also used for logging the ontology changes in the CHL is discussed later on.

Inconsistencies Detection

In this module ontology changes are resolved in a systematic manner to ensure that consistency of the ontology is not lost. The ontology may become inconsistent because of the changes. Two types of inconsistencies can occur, 1) syntactic: when an undefined entity at ontology or instance level is used, 2) semantic: when meaning of ontology entity is changed due to performed changes. To keep the ontology in consistent state deduced changes are introduced in the ontology. After this a complete request of both, the required changes and the deduced changes is made.

In the first version of the evolution management we are expecting expert intervention for resolving discrepancies, but machine learning techniques could be used to learn first from different types of change resolutions and then act for these conflicts in future without expert intervention. KAON API [5] is used to identify the alternatives and deduced changes. These changes are presented to the ontology engineer and then the ontology engineer selects one of the alternatives. For one induced change there may be more than one deduced changes. The decision is made in favor of those changes which have the lesser impact on the ontology structure and fewer deduce changes.

Change Implementation & Verification

All the induced and deduced changes which make a complete change request are applied to the ontology. The framework is designed to manage three characteristics. Firstly, when a change is applied then it should be completed in isolation, change must be atomic, durable, and consistent. Secondly, after every change implementation, change verification is made to check that the required changes have been committed to the ontology. And thirdly, after every change implementation the change must be logged in the change history log, to keep track of the changes performed in an ordered manner. This helps in undoing any changes by simply reversing the logged changes on the ontology.

Change History Log

It is a repository that keeps track of all the changes made to the ontology. It stores every change after it is implemented. The CHL is also required for reversibility purpose when an ontology engineer wants to undo or redo some of the changes then this log is accessed and changes are simply reverted. The log uses Jena based triple store and the change description provided by CHO to preserve the changes.

It stores a complete change set for changes with information like; who made these changes, reasons for changes, what kind (instant or interval) of change, if interval change then what is the beginning and end time of changes. It stores all types of changes that can happen to ontology during evolution, which are given in [9] and [11]. We consider composite changes as a set of atomic changes. CHO is used to enable the proper undo and redo operations which will completely revert back the whole change set and take the ontology to any previous state for undo operations.

We log all the changes with the time stamp at which they are performed. Though a single change is performed at an instant of time, at times a lot of changes (sequence of changes) are performed together over an extended time interval. To effectively log both types of changes we add time stamps using OWL-Time Ontology [8]. A single change is logged as a change at a time instant while a sequence of changes in combination is logged as change over a time interval. As the ontology can be target of change at different times triggered by ontology engineers located at distributed places, using a singleton time format (zone) might create some ambiguity or conflicts for certain changes. By the use of OWL-Time Ontology these problems/conflicts can easily be sorted out.

5. Implementation and Results

Though for the purpose of clarity and implementation of the claims, we have targeted "*Documentation*" ontology, but the proposed framework is not restricted to a specific domain.

Here we show the evolution process by an example. Fig 3(a) shows the new emerging concepts, while in Fig 3(b) we have the domain ontology O to which change will be made and Fig 3(c) shows the evolved ontology O. Due to space limitation we have shorten the description of the example.

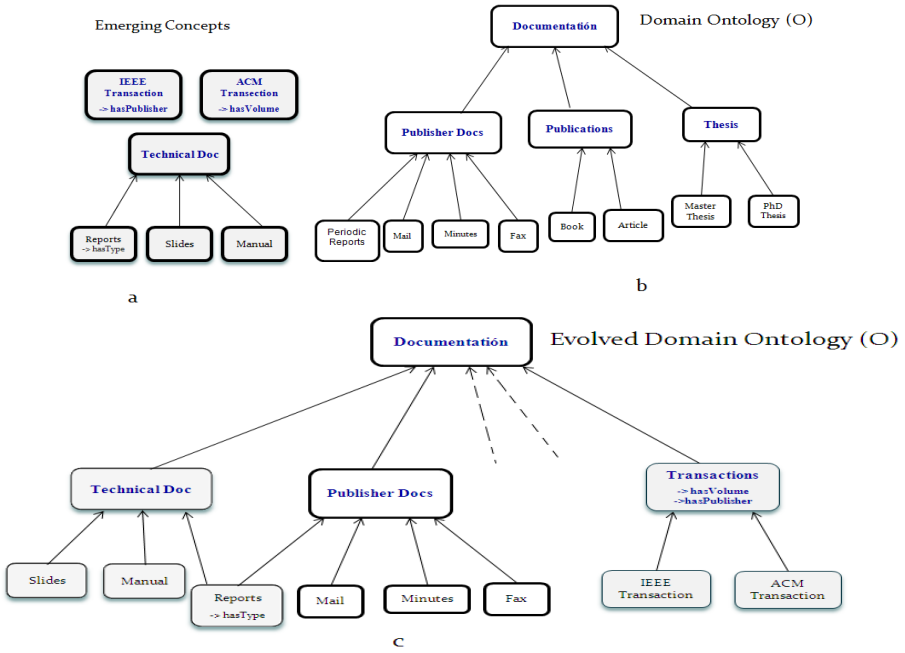

Fig 3. (a) Shows the emerging concepts, (b) domain ontology O and (c) domain ontology O after evolution

The first step in the process is detection and description of the changes. As described in Section *Change Detection & Description* Semantic Affinity is calculated using H-Match algorithm [20]. Semantic Affinity, $SA(c,c')$, between two concepts c and c' is a measure of their similarity. A strong match between the concept results in higher value of SA not greater than 1. On the other hand, 0 or smaller values stands for no match or weak similarity. For two concepts to be considered strongly similar their semantic affinity value shall be greater than the threshold ψ. Now the techniques discussed in [2], which are ontology matching and reasoning to find most suitable place for the new concept(s) are used, which suggest that IEEE Transaction and ACM Transaction come to be the sub concepts of Documentation concept so should be merged. For Technical Docs and its sub concepts are known to be the sub concept of the Documentation concept, where sub concept of Technical Docs i.e. Reports and Periodic Reports of Publisher Docs are same, so merge it and make Reports as sub concept of both the concepts. Now these changes are represented in proper format. Descriptions of some of the changes are given in Fig 4, where these are represented using N3 notation.

In the Inconsistencies Detection module, consistency of an ontology for the required changes is checked. It is detected using an external source (WordNet) that IEEE Transaction and ACM Transaction are disjoint concepts so they must be separated. This inconsistency is resolved through deduced changes which are incorporated to handle inconsistency. Then a complete change request of both, the required and the deduced changes are made as shown in Fig 4(b).

In the Implementation and Verification module, all the change request changes are made to ontology O. Verification is for two purposes:

- The consistency of ontology O is checked and verified after every change made, because still there is a chance of inconstancies. If so, then for its resolution a request is made back to inconsistency detection module.
- All the changes implemented to the ontology O are then logged in change log, to be able to keep track of implemented changes and to provide undo and redo facility. In verification phase it is checked that whether all the required changes are made or not by accessing the change request.

a

```
default:Class_Addition_1
a       default:Class_Addition;
default:hasChangeTarget
<http://www.niit.edu.pk/Documentation.owl#ACM_IEEE_Transaction>;
default:isSubClassOf
<http://www.niit.edu.pk/Documentation.owl#Documentation>.

default:Property_Addition_1
a       default:Property_Addition;
default:hasChangeTarget
<http://www.niit.edu.pk/Documentation.owl#hasPublisher>

default:Property_Addition_2
a       default:Property_Addition ;
default:hasChangeTarget
<http://www.niit.edu.pk/Documentation.owl#hasVolume>
```

b

```
default:Class_Addition_1
a       default:Class_Addition;
default:hasChangeTarget
<http://www.niit.edu.pk/Documentation.owl#Transaction>;
default:isSubClassOf
<http://www.niit.edu.pk/Documentation.owl#Documentation>.

default:Property_Addition_1
a       default:Property_Addition;
default:hasChangeTarget
<http://www.niit.edu.pk/Documentation.owl#hasPublisher>

default:Property_Addition_2
a       default:Property_Addition;
default:hasChangeTarget
<http://www.niit.edu.pk/Documentation.owl#hasVolume>

default:Class_Addition_9
a       default:Class_Addition;
default:hasChangeTarget
<http://www.niit.edu.pk/Documentation.owl# ACM Transaction>;
default:isSubClassOf
<http://www.niit.edu.pk/Documentation.owl#Transaction>.

default:DisjointClass_Addition_1
a       default:DisjointClass_Addition ;
default:hasChangeTarget
<http://www.niit.edu.pk/Documentation.owl#IEEE Transaction>;
default:isDisjointWith
<http://www.niit.edu.pk/Documentation.owl#ACM Transaction>;
default:isSubClassOf
        <http://www.niit.edu.pk/Documentation.owl#Transaction>.
```

Fig 4. (a) Change representation using CHO, and (b) change representation using CHO with deduced changes.

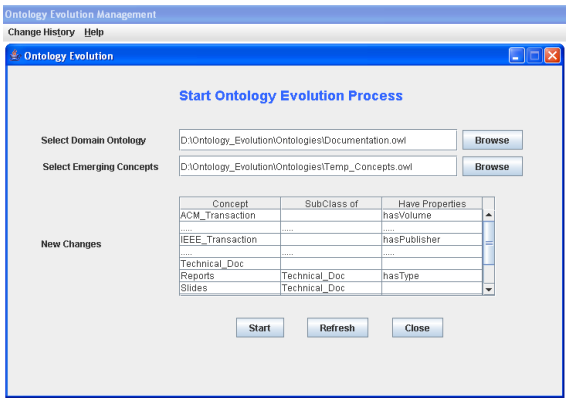

Fig 5. Snapshot of prototype system.

Fig 5 is a snapshot of the prototype underdevelopment. In this the domain ontology is selected to which the changes are to be made then the new emerging concepts are selected, which on selection are also populated into table given below. From here on the ontology evolution process starts which includes first matching and then accommodating the new changes in proper places.

Although the prototype system needs some human intervention for the changes to be made to the ontology, but this expert dependency is expected to be eliminated once the system is completely developed.

6. Conclusions and Step Ahead

Ontology evolution is not a new area of research; the work which is conducted in this field is incorporating work from other related fields such as ontology matching, merging, integration, and reasoning. In this paper we presented a framework for ontology evolution. We first introduced the change detection and description module for new changes as new concepts emerge in the domain, and then represent these identified changes in a consistent format using Change History Ontology which acts as a glue to bind different components in the framework.

New changes are checked for ontology consistency and where required, deduced changes are embedded. The changes are then implemented and stored in the Change History Log. Lastly the ontology is checked and verified for the required changes. As ontology changes are stored in Change History Log, these logged changes can also be used for deduced changes, change prediction, and visualization of change effects on ontology.

7. References

[1] S. Castano, A. Ferrara, G. Hess, "Discovery-Driven Ontology Evolution". *The Semantic Web Applications and Perspectives (SWAP),* 3rd Italian Semantic Web Workshop, PISA, Italy, 18-20 December, 2006.

[2] S. Castano, A. Ferrara, and S. Montanelli, "Evolving open and independent ontologies," *Journal of Metadata, Semantics and Ontologies (IJMSO),* vol. 1, No.4 pp. 235 - 249, 2006.

[3] S. Castano, A. Ferrara, and S. Montanelli. "Matching ontologies in open networked systems". *Techniques and applications, Journal on Data Semantics (JoDS),* vol. V, pp. 25-63, 2006.

[4] G. Flouris, D. Plexousakis, G. Antoniou, "A Classification of Ontology Changes", *In the Poster Session of Semantic Web Applications and Perspectives (SWAP),* 3rd Italian Semantic Web Workshop, PISA, Italy, 2006.

[5] T. Gabel, Y. Sure, and J. Voelker. "*KAON – ontology management infrastructur*e". D3.1.1.a, SEKT Project: Semantically Enabled Knowledge Technologies, March 2004.

[6] P. Haase, L. Stojanovic, "Consistent Evolution of OWL Ontologies". *In Proceedings of the 2nd European Semantic Web Conference (ESWC),* 2005.

[7] P. Haase, Y. Sure. "*State of the Art on Ontology Evolution*", D3.1.1.b, SEKT Project: Semantically Enabled Knowledge Technologies, August 2004.

[8] J. R. Hobbs, and F. Pan, "An Ontology of Time for the Semantic Web". *ACM Transactions on Asian Language Processing (TALIP*) vol. 3, No. 1, pp. 66-85, 2004.

[9] M. Klein. "*Change Management for Distributed Ontologies"*. PhD Thesis, Department of Computer Science, Vrije University, Amsterdam, 2004.

[10] M. Klein and N. F. Noy, "A component-based framework for ontology evolution", *In Proceedings of the (IJCAI-03) Workshop on Ontologies and Distributed Systems, CEUR-WS*, vol. 71, 2003.

[11] Y. D. Liang, "*Enabling Active Ontology Change Management within Semantic Web-based Applications*". Mini PhD Thesis, University of Southampton, 2006.

[12] N. F. Noy and M. Klein, "Ontology evolution: Not the same as schema evolution", *Knowledge and Information System,* vol. 6, no. 4, pp. 428–440, 2004.

[13] N. F. Noy, S. Kunnatur, M. Klein, and M. A. Musen, "Tracking changes during ontology evolution". *In Proceeding of the 3rd International Semantic Web Conference (ISWC),* Hiroshima, Japan, November 2004.

[14] N. F. Noy and M. Klein, "Tracking Complex Changes During Ontology Evolution". *Second International Semantic Web Conference (ISWC),* Sanibel Island, Florida, 2003.

[15] D. Oberle, R. Volz, B. Motik, and S. Staab, "*An extensible ontology software environment*", In Handbook on Ontologies (Series of International Handbooks on Information Systems), pp. 311–333, Springer, 2004.

[16] P. Plessers, and O. De. Troyer, "Ontology change detection using a versioning log". *In Proceeding of the 4th International Semantic Web Conference (ISWC)*, Galway, Ireland, pp. 578–592, 2005.

[17] L. Stojanovic, A. Madche, B. Motik, and N. Stojanovic, "User-driven ontology evolution management," *In European Conference on Knowledge Engineering and Management (EKAW),* pp. 285-300, 2002.

[18] L. Stojanovic, "*Methods and Tools for Ontology Evolution*". PhD thesis, University of Karlsruhe, 2004.

[19] L. Stojanovic and B. Motik. "Ontology evolution within ontology editors". *In Proceedings of the OntoWeb-SIG3 Workshop at the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW),* pp. 53–62, September 2002.

[20] M. Tury, M. Bielikova. "An Approach to Detection Ontology Changes". *First international workshop on adaptation and evolution in web systems engineering (AEWSE)*, 20006.

[21] Asad Masood Khattak, Khalid Latif, Sharifullah Khan, Nabeel Ahmed, "Managing Change History in Web Ontologies," *Semantics, Knowledge and Grid, International Conference on*, pp. 347-350, 2008 Fourth International Conference on Semantics, Knowledge and Grid, 2008.