After Effects of Ontology Evolution

Asad Masood Khattak, Zeeshan Pervez, Sungyoung Lee, Young-Koo Lee

Department of Computer Engineering, Kyung Hee University, Korea

{asad.masood, zeeshan, sylee}@oslab.khu.ac.kr, yklee@khu.ac.kr

***Abstract*— Use of ontology in Information Systems and Knowledge Sharing Systems are increasing that gives more importance to proper maintenance of these ontologies in use. Ontology change management is a collaborative process that incorporate areas like; ontology engineering, evolution versioning, merging, integration, and maintenance. As experts develop better understanding of the domain, incorporate changes accordingly to the body of knowledge, as a result the body of knowledge evolves from one state to another. Preserving consistency while accommodating changes is a crucial task that needs special attention. In this paper we provide a brief review of state of the art in the field of ontology evolution that set the stage for the unfolded challenges in ontology evolution to complete the process automatically. Then we also discuss very important issues that need special attention to minimize the after effects of ontology evolution.**

I. INTRODUCTION

Use of ontology is increasing in Information Systems and Knowledge Sharing Systems that increase the significance of ontology maintenance. *Ontologies are formal description of shared conceptualization of a domain of discourse*. Ontologies are complex in nature and often large structured, their development and maintenance incorporates research areas like; engineering, evolution, versioning, merging, and integration [3]. For better system accuracy and performance, up-to-date and complete information must be maintained in the knowledgebase. The domain knowledge evolves as the communities of practices concerned with knowledge develop better understanding of their perceived knowledge [21]. Ontology evolution is the change in expert perception about the domain in view [17]. The evolution process is the ontology modification process that captures and accommodates new information in the domain knowledge.

Ontology evolves from one consistent state to another [6] and to accomplish the evolution process several different sub tasks are involved [10] i.e. *capture change, change representation, semantics of change, change implementing and verification, and change propagation.*

Research on ontology evolution is being carried out by different researcher groups, and they have somehow overlapping approach towards the solution to the problem. These approaches do have some pragmatic advantages and disadvantages. The current ontology evolution techniques have several weaknesses, such as: specifications of new changes, resolving inconsistencies (selecting deduced changes from available alternatives), and also undo and redo in case we want to recover the ontology are all manual [11]. In order to automate the process of ontology evolution, we need to automate all the above discussed tasks, where this automation is important because human intervention is time consuming and error prone.

The goal of this research is to provide a brief review of approaches followed by different researchers groups for ontology evolution. We highlight the main features of the all the approaches using a summary table that describe different approaches with their contributions. Then these approaches are critically analysed. We also discuss some open problems that need to be addressed in order to completely automate the evolution process. We discuss some consequent issues/effects of ontology evolution on dependent data, applications, and artefacts and suggest possible solutions for these issues to eliminate and/or minimize the after ontology evolution effects.

This paper is arranged as follows: Section II briefly discusses the change and change management activities. Section III presents different ontology evolution approaches proposed by different researchers. In Section IV we present the challenges still needs to be worked out for complete automation of evolution procedure and minimize the after effects of ontology evolution. Finally we conclude our discussion in Section V.

II. ONTOLOGY CHANGE MANAGEMENT ACTIVITIES

Most of ontology change management activities are discussed in [12]. These activities are related and in some cases they are overlapping but fundamentally these are different activities such as: *Ontology Engineering, Ontology Evolution, Ontology Segmentation, Ontology Versioning, Ontology Merging, Ontology Integration, and Ontology population.* Focus of this research work is on the evolution part of ontology change management and how to automate the evolution process and minimize the after evolution effects of ontology on information systems. The evolution in ontology is mainly of two types i.e. *Ontology Population* and *Ontology Enrichment* [12]. Different changes have different affects on overall ontology and most of these changes are discussed in [1 and 14].

A single change in ontology can be of both simple and complex nature depending upon resources that are affected with it. Changes like *New Concept Change, Concept Hierarchy Change, Concept with Changed Properties, Concept with Changed Restrictions, Simple vs. Aggregated Concept,* and *Concept vs. Property* are discussed in [12]. Understanding of change types (i.e. simple and complex) is

978-1-4244-6949-9/10/$26.00 ©2010 IEEE

necessary to correctly handle explicit and implicit change requirements [5], and consequently understand the effects of these changes on ontology and ontology based information systems. The details of different issues and their possible solutions are discussed later. For instance a simple change that is deletion of class from a hierarchy can result in lot of different complex changes (see Figure 1). Now suppose that all the subclasses have their instances and a simple change of class (i.e. Lecturer) deletion occurred. (1) This deletion needs some decisions like weather all the instances of Lecturer should also be deleted and is loss of information. If not then how these instances should be maintained in the hierarchy. (2) Consider all these subclasses are disjoint and in this case, instances of Lecturer cannot be distributed among the disjoint classes. (3) Consistency of ontology after this change is not guaranteed.

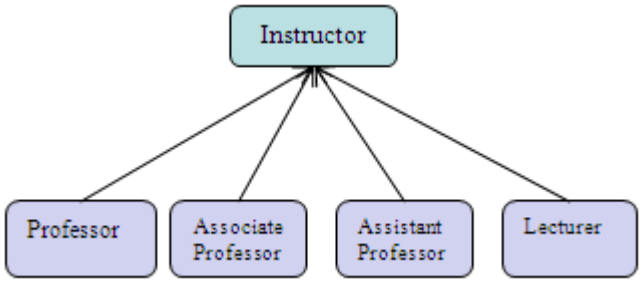

Figure 1, Class hierarchy for a simple change

III. Ontology Evolution Approaches

Ontology over time needs to be updated to accommodate changes in the domain, user requirements, and to incorporate incremental improvement in the system. In this section we will briefly discuss different ontology evolution approaches with reference to their comparison in Table 1. At the end we will critically analyse these approaches.

Initially, L. Stojanovic and B. Motik in [21] talked about the support different ontology editors provides in order to develop an ontology, and also discussed the limitations, complexities, and usability issues of these tools for ontology evolution management. As ontology developed for any domain needs to be refined, so for these refinements we have to make some changes in ontology and update it. Therefore, the methods for ontology evolution to cope with the changes systematically are an essential requirement for ontology editors.

Different requirements that an ontology management system (editor) should provide for ontology evolution and propagation of changes to dependent data and applications are discussed in [21]. They first provided the functional requirements for the system to properly interact with the underlying model and also provided multiple types of changes (related to Class, Properties, hierarchy, instances, and restrictions) that can take place during the process of evolution. In [4, 13, 14, 16, and 22], the new changes are specified by ontology engineer, [18, 19, and 20] detect new changes among two different versions of ontologies using PromptDiff (Protégé plug-in), OntoView [8], and H-Match [2] algorithms. The systems discussed in [1, 10, and 22] detect new changes mostly using WordNet and H-Match [2] for finding correspondence among new changes and already existing concept(s) in the source ontology. These changes are then represented as a complete change request in formal representational format developed by different researchers such as; Change and Annotation Ontology (CHAO) [14], Change Log [20], and Change History Ontology (CHO) [9]. Resolving the conflicts (inconsistencies) due to new changes is one of the most focused issues of ontology evolution algorithm. In most of the systems (as given in Table 1) ontology engineer resolve these conflicts, but some systems like KAON [4] use a predefined resolution strategy, and Evolva of NeOn Toolkit [22] uses a customized run time strategy for conflict resolution.

After implementation of the new changes, some of the systems [4, 10, 14, and 20] provide the facility for change logging for the purpose of undo/redo and ontology recovery (both roll-back and roll-forward) [11]. After validation of implemented changes, the changes are propagated to the dependent data, application, and artefacts using push-based and pull-based [21] approaches. But most of the evolution systems are not providing the facility of propagation as they follow the Semantic Web context, where it is highly possible that the ontology might be in use of unknown dependent (by definition, ontology is shared among community of users) and they might not need the updates. So, in this case, the pull-based approach is suitable and pull-based approach exclude change propagation phase from ontology evolution procedure.

A. Critical Analysis

Existing ontology evolution systems do not consider new emerging concept(s), main reason for semi-automated evolution procedure. In [21] the authors talk about different requirements needs to be fulfilled in order to achieve ontology evolution properly, but they do not provide any tangible results. In [19 and 21], the users manually create the requests for changes, the conflicts are manually resolved by experts. In [1], author focused on discovery of new change (resource) and afterwards ontology expert insert the resource at suitable place suggested by the system.

TABLE1. SUMMARY OF ONTOLOGY EVOLUTION APPROACHES. THE LAST COLUMN REPRESENTS SYSTEM MATURITY LEVEL.

Approaches	Change Request	Change Representation	Conflict Resolution	Change Implementation	Change Propagation	System Working
L. Stojanovic, et al. [21].	The complete change request is represented in formal representational format. These changes (due to business requirements are specified by ontology engineer.		Ontology engineer resolve all the inconsistencies due to requested changes by incorporating deduced changes.	The requested changes (including deduced changes) are applied to the source ontology.	Applied changes are propagated to dependent data, applications, and ontologies. Out-of-date instances are simply replaced with the up-to-date instances.	User Dependent
M. Klein, N. Noy, et al. [13, 14, 16]	Specified by ontology engineer.	Developed Change and Annotation Ontology (CHAO) to represent change request.	Ontology engineer involvement.	Suggested that tools should provide interface for user interaction.	Consistent propagation of changes to distributed instances of ontology.	User Dependent
T. Gabel, et al. [4] (KAON)	Specified by ontology engineer	Formal representation of changes	Use predefined strategies for conflict resolution and avoiding side-effects.	Provide interface for user interaction and also log the changes.	Propagation of changes to dependent artefacts.	Semi automatic
P. Plessers, et al. [19]	Different versions of ontologies are used in this approach. The changes among different versions are represented formally.		After changes implementation, it checks for inconsistencies and if present then it makes the change recovery.	First it implements the change request and then check for any conflicts.	It does not support change propagation as it works on versions.	User dependent
D. Oberle, et al. [18, 20]	Changes are detected among two versions by using PromptDiff and OntoView [8], and compile a complete change request	Formally represented using their developed semantic structure.	Ontology engineer resolve inconsistencies by introducing deduced changes	With changes implementation, all the changes are also logged for undo/redo purpose	It does not support change propagation as it works on versions	User dependent
S. Castano, et al. [1]	Changes are recognized automatically by analysing domain artefacts. H-Match [2] and WordNet [1] are used for new change detection.	Changes are then formally represented.	Inconsistencies are resolved by ontology engineer.	Changes are made by ontology engineer.	Change propagation is not focused.	Semi automatic
A. M. Khattak, et al. [10]	New changes such as (change in single concept, group of concepts and concepts in a hierarchical structure) are detected automatically using H-Match [2] and WordNet. Change representation is provided by Change History Ontology (CHO) [9].		For conflict resolution KAON API [4] is used with some suggested extensions.	Changes are implemented atomically and after every change implementation, these are logged in CHL [9]. At the end all changes are validated against the change request.	Change propagation is not handled in this approach.	Suggestions toward automation
F. Zablith [22]	Changes can be specified by user and detected automatically. They also use WordNet for new change detection. Then these changes are formally represented using different representation techniques followed in their overall NeOn Toolkit.		A new developed algorithm for conflict resolution strategy is partially implemented.	Changes are implemented and verified.	Change propagation is the focus for 2nd phase with conflict resolution.	Towards automatic ontology evolution

[1] http://wordnet.princeton.edu/wordnet/download/

The main concerns in [10 and 22] systems are the best matching resource(s) selection for the newly emerging change (resource). Inconsistency resolution is the most critical problem that not only needs attention during evolution but also the evolution procedure. The consistency is checked for; consistency of ontology with the system putting queries on the ontology, consistency with the other side matching ontology, and consistency with the business rules of organization. In [10] we proposed a training process for different deduced changes but during evolution process. Training a system for different changes and their consequent deduced changes is a tough job, and even after proper training the system results may not be according to user intentions. For a single conflict there might be many alternative deduced changes, and deciding upon a certain change is in itself another issue [22].

IV. Open Challenges

Here we briefly explain some of the challenges that need to be solved for the purpose of achieving automated ontology evolution procedure [12]. Then we discuss in detail some of the after effects of ontology on the dependent systems, data, and other ontologies. We also suggest some possible solutions for these challenges to eliminate and/or even minimize the after evolution effects.

A. New Change Detection

In newly emerging concept(s) (single concept, group of concepts, and concepts in a hierarchical structure), when a change is detected then matching [2 and 18] is applied to find out correspondence among new concepts and already existing concepts. This helps in proper insertion of the new concept in concept hierarchy. But still there are two different problems; (1) *Relevance Detection* is that how much is the new concept related to existing concepts. (2) *Selection among newly Detected Changes:* it is quite possible that more than one concept is related to the newly detected concept, now the issue is that which alternative should be selected, for detail please refer to [12].

B. Conflict Detection

To make ontology consistent after introducing changes is the most critical issue. For consistency, deduced changes are introduced in order to resolve conflicts. Still most of the existing system use expert intervention for resolving the conflicts [1, 13, 19, and 21]. For detail on this issue please refer to [12].

C. Query Reformulation

Query written over one schema does not give correct results when executed over another schema, so needs to be reformulated in order to fulfil the schema requirements. Same is true for ontology, so when ontology evolves form one consistent state to another then the query written over previous state needs to be reformulated to extract required results from the ontology [15]. The author in [15] proposed a five phase query reformulation procedure for evolved ontologies. The main modules of the procedure are, *capture*, *instantiate*, *analyse*, *update*, and *respond* (for details please refer to Y. D. Liang Mini thesis [15]). The evaluation of the system is done using two different versions of CRM[2] ontology. The main idea behind this work is to maintain the changes that are reason for evolution in a repository and later used them for query reformulation.

```
<Change_Person rdf:ID="Change_Person_Instance_2009_07_06_16_31_50">
    <hasAuthorName rdf:datatype="&xsd;string">
            Asad Masood Khattak</hasAuthorName>
</Change_Person>
<owl:Class rdf:ID="Change_Set">
    <rdfs:subClassOf rdf:resource="#Ontology_Change"/>
</owl:Class>
<Change_Set rdf:ID="Change_Set_Instance_2009_07_06_16_31_50">
    <hasOntology rdf:datatype="&xsd;string">uni1</hasOntology>
    <hasChangeBeginTime rdf:datatype="&xsd;string">
     2009-07-06T16:31:50</hasChangeBeginTime>
    <hasChangeReason rdf:datatype="&xsd;string">
     Un-Wantedly Deletion</hasChangeReason>
  <hasChangeAuthor rdf:resource="#Change_Person_Instance_2009_07_06_16_31_50"/>
</Change_Set>
<Class_Addition rdf:ID="Class_Addition_Instance_1246865528281">
    <hasTimeStamp rdf:datatype="&xsd;string">1246865528281</hasTimeStamp>
    <hasChangedTarget/>
    <isSubClassOf/>
    <isPartOf rdf:resource="#Change_Set_Instance_2009_07_06_16_31_50"/>
</Class_Addition>
<Class_Addition rdf:ID="Class_Addition_Instance_1246865554156">
    <hasTimeStamp rdf:datatype="&xsd;string">1246865554156</hasTimeStamp>
    <hasChangedTarget/>
    <isSubClassOf/>
    <isPartOf rdf:resource="#Change_Set_Instance_2009_07_06_16_31_50"/>
</Class_Addition>
.....
.....
.....
<Class_Renaming rdf:ID="Class_Renaming_Instance_1246865547906">
    <hasTimeStamp rdf:datatype="&xsd;string">1246865547906</hasTimeStamp>
    <hasOldName rdf:datatype="&xsd;string">Class_1</hasOldName>
    <hasChangedName rdf:datatype="&xsd;string">MS_Student</hasChangedName>
    <isSubClassOf/>
    <isPartOf rdf:resource="#Change_Set_Instance_2009_07_06_16_31_50"/>
</Class_Renaming>
<Class_Renaming rdf:ID="Class_Renaming_Instance_1246865561718">
    <hasTimeStamp rdf:datatype="&xsd;string">1246865561718</hasTimeStamp>
    <hasOldName rdf:datatype="&xsd;string">Class_2</hasOldName>
    <hasChangedName rdf:datatype="&xsd;string">PhD_Student</hasChangedName>
    <isSubClassOf/>
    <isPartOf rdf:resource="#Change_Set_Instance_2009_07_06_16_31_50"/>
</Class_Renaming>
<Class_Renaming rdf:ID="Class_Renaming_Instance_1246865595718">
    <hasTimeStamp rdf:datatype="&xsd;string">1246865595718</hasTimeStamp>
    <hasOldName rdf:datatype="&xsd;string">Class_4</hasOldName>
    <hasChangedName rdf:datatype="&xsd;string">Professor</hasChangedName>
    <isSubClassOf/>
    <isPartOf rdf:resource="#Change_Set_Instance_2009_07_06_16_31_50"/>
</Class_Renaming>
```

Figure 2, Representation of *Change_Set* instance with corresponding change entries, reason for O_1 (given in Figure 3) evolution to O_1' using *CHO [9]*

One of the limitations of this system is that it was only tested over two specific versions of CRM ontology, so its scalability is a question mark, not only for other ontologies but also for different versions of CRM ontology. Secondly, the structure for logging the ontology changes is also not suitable for query reformulation over more than two ontologies at the same time as its hard to extract the changes from the log that correspond to a particular state of ontology. In [9], a *Change History Ontology (CHO)* is presented that logs all the ontology changes in atomic manner and also keep the changes separate from those that correspond to different state of ontology. The notion of *Change_Set* has been

[2] http://cidoc.ics.forth.gr/index.html

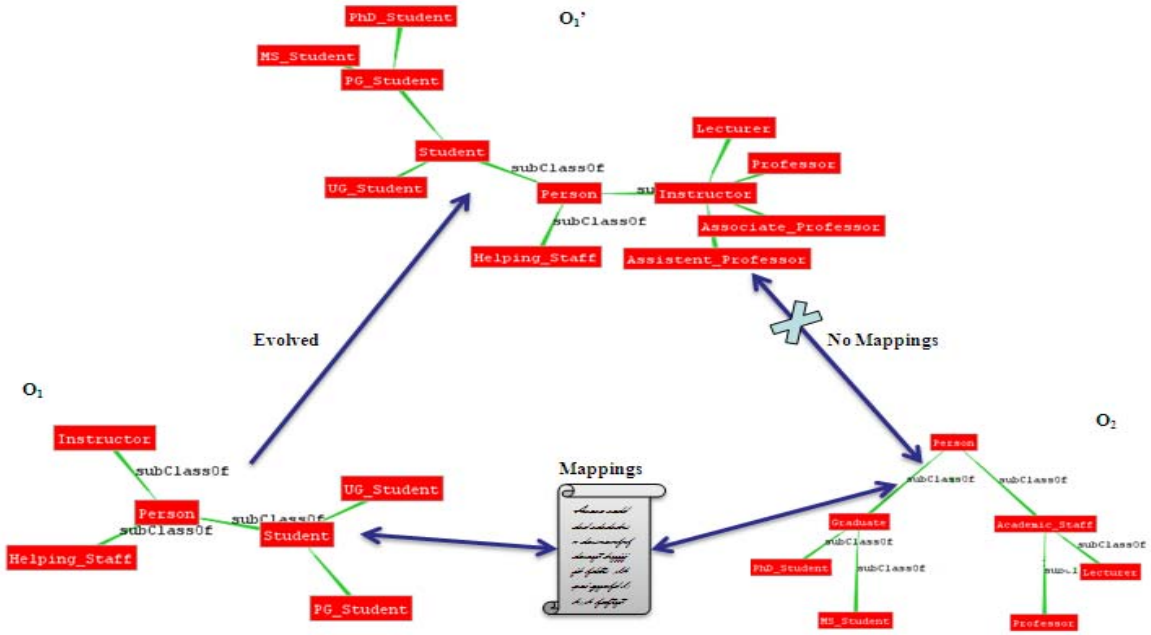

Figure 3, Ontology O_1 and O_2 having mappings, Ontology O_1 have evolved from state O_1 to state O_1', so the previous mappings are no more reliable as there are different changes introduced in O_1'.

introduced that bundles all the ontology changes together that are the cause of evolution from one state to another. So this separate *Change_Set* instances helps in proper reformulation of query for required version/state of ontology (Figure 2 is an example of *Change_Set* instance with the corresponding changes that cause the ontology O_1 evolved to O_1' state as shown in Figure 3).

D. Rebuilding Ontology Mappings

Mappings among ontologies are required to translate query and/or share information [15 and 7]. When ontology evolves from one consistent state to another then its mapping with the other ontologies are no more reliable and query execution over such mappings will produce unpredictable results. So when an ontology having mappings established with other ontologies evolved then there is also a need for re-establishment of these mappings.

There is no such solution for reengineering the broken mappings among the evolved ontologies except to completely re-establish the mappings. To re-establish the mappings among small ontologies is not an issue, but if ontologies like Google Classification [3], Wiki Classification [4], ACM Classification Hierarchy[5], and MSC Classification Hierarchy[6] evolve with very little changes, then complete re-establishment of mappings among such ontologies is a time consuming process. To solve this problem in time efficient manner, we propose that *Change History Log (CHL)* [9] containing all the changes (reason for ontology evolution), can play an important role here. It actually can answer to two situations;

- When two or more ontologies are to be mapped for the first time, then the change entries from *CHL* can play a vital role. For example to align (map) a class C_{11} from ontology O_1 to a class C_{21} in ontology O_2 but in ontology O_2 match is not found. In this case *CHL* can be consulted which will give information about all the classes that might have changed in recent past, and can have some type of match (relationship) with a class C_{11} from ontology O_1.
- Secondly, if there are mappings already established between two ontologies O_1 and O_2 (see Figure 3). So they can share information and reformulate queries using these mappings. Now let's consider that one ontology i.e. O_1 has evolved to another state O_1'. So, there are no mappings between ontology states O_1' and ontology O_2, so they cannot share information. In such case to save time in re-establishing mappings, we can use the change history record from the *CHL* and use these change entries to reconcile the mapping between evolved ontologies rather than considering the complete ontology. This reconciliation is also applicable if both the ontologies evolve.

E. Change Traceability

Change History Log (*CHL*) stores all the changes in formal manner provided by Change History Ontology [9]. Changes of specific time interval is logged as one *Change_Set* as shown in Figure 2, and this *Change_Set* changes are the reason for ontology evolution. Now this provides us the facility to revert these changes back on the ontology to get the

[3] http://www.google.com/Top/Reference/Libraries/Library_and_Information_Science/Technical_Services/Cataloguing/Classification/
[4] http://en.wikipedia.org/wiki/Taxonomic_classification
[5] http://www.acm.org/about/class/1998/
[6] http://www.math.niu.edu/~rusin/known-math/index/index.html

previous consistent state of ontology. These stored changes not only provide the facility for rollback, but is also used for roll-forward operations based on user request.

Managing ontology changes during evolution in *CHL* is also helpful for new user to get the understanding of the changes made to ontology. Using entries of *CHL* one can also extract/understand the change in semantics of changed conecpt(s). Annotation can also be added with all the changes like; reason for the change, effects of the change on dependent data, application, and other artefacts do help in understanding the changes in ontology, data, and application. As all the changes are logged and properly managed, so one can also deduce some pattern by applying some mining algorithms which can help in next change predictions.

The ontology changes (logged) can be used to visualize the change effects on ontology in different states through which it passed to the current state. So this visualization of change effects on ontology will provide the facility to temporally trace the ontology changes and better understand its evolution behaviour.

V. Conclusions

Ontology evolution incorporate work from related areas like ontology matching, merging, integration, versioning, and reasoning. We talked about the changes and different approaches followed for ontology evolution with their comparative analysis. We also briefly discussed some open challenges still unhandled. Some of the after effects of ontology evolution on ontology and on Information Systems based on ontology with suggested solutions to these after effects are also discussed in detail.

Currently, we are working on reconciliation of ontology mapping and change predictions in dynamic ontologies.

VI. Acknowledgement

This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency)" (NIPA-2009-(C1090-0902-0002)). This work was supported by the IT R&D program of MKE/KEIT. [2009-S-033-01, Development of SaaS Platform for S/W Service of Small and Medium sized Enterprises].

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. “Matching ontologies in open networked systems”. Techniques and applications, Journal on Data Semantics (JoDS), vol. V, pp. 25-63, 2006.

[3] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, G. Antoniou. Ontology Change: Classification and Survey. Knowledge Engineering Review (KER), 23(2), pages 117-152, 2008.

[4] T. Gabel, Y. Sure, and J. Voelker. “KAON – ontology management infrastructure”. D3.1.1.a, SEKT Project: Semantically Enabled Knowledge Technologies, March 2004.

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

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

[7] W. Hu and Y. Qu. “Falcon-AO: A Practical Ontology Matching System”, Journal of Web Semantics, 2007.

[8] M. Klein, A. Kiryakov, D. Ognyanov and D. Fensel, "Finding and characterizing changes in ontologies", 21st International Conference on Conceptual Modeling, International Conference on Conceptual Modeling, Tampere, Finland, pp. 79-89, 2002.

[9] A. M. Khattak, K. Latif, S. Khan, N. 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.

[10] A. M. Khattak, K. Latif, S. Y. Lee, Y. K. Lee, and T. Rasheed, “Building an Integrated Framework for Ontology Evolution Management”,12th Conference on Creating Global Economies through Innovation and Knowledge Management (IBIMA), Malaysia, June 2009.

[11] A. M. Khattak, K. Latif, S. Y. Lee, Y. K. Lee, M. Han, and H. Il-Kim, “Change Tracer: Tracking Changes in Web Ontologies”,21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Newark, USA, November 2009.

[12] A. M. Khattak, K. Latif, S. Y. Lee, Y. K. Lee, “Ontology Evolution: A Survey and Future Challenges”, The 2nd International Conference on u- and e- Service, Science and Technology (UNESST 09), Jeju, Korea, December 10 ~ 12, 2009.

[13] 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.

[14] M. Klein. “Change Management for Distributed Ontologies”. PhD Thesis, Department of Computer Science, Vrije University, Amsterdam, 2004.

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

[16] N. F. Noy, A. Chugh, W. Liu, M. A. Musen, “A Framework for Ontology Evolution in Collaborative Environments”, International Semantic Web Conference – ISWC, pp. 544-558, 2006.

[17] 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.

[18] 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.

[19] 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.

[20] D. Rogozan, and G. Paquette. “Managing Ontology Changes on the Semantic Web”. IEEE/WIC/ACM International Conference on Web Intelligence. 2005.

[21] 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.

[22] F. Zablith, Ontology Evolution: A Practical Approach, Poster at Proceedings of Workshop on Matching and Meaning at Artificial Intelligence and Simulation of Behaviour (AISB), Edinburgh, UK, 2009.