2009 21st IEEE International Conference on Tools with Artificial Intelligence

Change Tracer: Tracking Changes in Web Ontologies

Asad Masood Khattak[1], Khalid Latif[2], Manhyung Han[1], Sungyoung Lee[1], Young-Koo Lee[1], Hyoung-Il Kim[3]

[1] *Dept of Computer Engineering, Kyung Hee University, Korea,*
{asad.masood, smiley, sylee}@oslab.ac.kr, yklee@khu.ac.kr

[2] *School of Electrical Engineering and Computer Science, NUST, Pakistan*
khalid.latif@seecs.edu.pk

[3]*Digital Media & Communications R&D Center, Samsung Electronics Co. Ltd.*
meeso.kim@samsung.com

Abstract

Knowledge constantly grows in scientific discourse and is revised over time by domain experts. The body of knowledge will get structured and refined as the Communities of Practice concerned with the field of knowledge develop a deeper understanding of issues. The knowledge model, as a result evolves to a new state to accommodate the new knowledge. Keeping trail of these changes in semantically rich and formally sound mechanism, has pragmatic advantages for providing the undo and redo facility and recover to a previous state of the knowledge body (i.e. ontology). In this research, we have developed and tested comprehensive methodological framework for Change Tracer. The ontology changes are captured and then stored in Change History Log (CHL) in conformance to Change History Ontology (CHO). The CHL is later used for reverting ontology to a previous consistent state and visualization of change effects on ontology. The system is compared with ChangesTab of Protégé, a comprehensive evaluation of the accuracy of roll-back and roll-forward algorithm has been conducted over Documentation ontology. The system is also tested over a standard dataset of OMV and high accuracy results are observed for both roll-back and roll-forward algorithms.

1. Introduction

O*ntologies are formal description of shared conceptualization of a domain of discourse.* Ontology change management is the solution to the problem of deciding the modifications to perform in ontology in response to a certain need for change [1]. It also deals with the implementation of these changes and the management of their effects in depending data, ontologies, services, applications, and agents. Ontology evolution process deals with the growth of ontology in response to certain need for change or the prospective under which the domain is viewed has changed [2, 3].

Ontology change management is a complicated and multifaceted task, which has led to the emergence of several different, but closely related, research areas. Ontology Integration, Merging, Versioning, and Evolution deal with different aspects of this problem [1]. Changes do occur in ontology and are reflected in the ontology by implementing these changes. As a result it evolves to a new state [4, 2]. Consequently, an ontology change management solution has to answer a number of questions [5]. First question is posed to the systems' overall working, "how to maintain all the changes in a consistent and coherent manner?" Other revolves around the applications of all these logged changes for the purpose of ontology recovery, visualization of change effects and understanding for the semantics of change.

The goal of this research article is to provide preliminary experimental results for our semantic structure and framework [5, 6] for temporal traceability in ontology evolution management. We developed Change History Ontology (CHO) [5] for maintaining ontology changes semantically. We envisioned a number of applications for the logged changes such as, ontology change management, change in semantics of the concepts, ontology recovery in case the system crashes, query reformulation, reconciliation of ontology mappings, change traceability, and to some extent navigation

1082-3409/09 $26.00 © 2009 IEEE
DOI 10.1109/ICTAI.2009.42

and visualization of the changes and change effects [6].

We have implemented and build a framework as a plug-in for Protégé (an ontology editor) as '*Change Tracer*'. It automatically detects and logs all the changes happened to ontology using CHO, triggered by the change request from ontology engineer. After that, whenever required, the CHL changes are accessed. The plug-in roll-back and roll-forward any changes and get the ontology in any previous consistent state [6]. We have tested its working on *Documentation* ontology and the experimental results depict excellent performance of the system in terms of roll-back and roll-forward of ontology changes. We have also compared our system results for change capturing with *ChangesTab* of protégé and our system has outperformed *ChangesTab*. We have also tested *Change Tracer* working over a standard dataset OMV and results depicts excellent performance of the system in terms of roll-back and roll-forward of ontology changes.

This paper is arranged as follows: Section 2 discusses the related work. Section 3 is an introduction to *Change Tracer* framework and CHO. Section 4 comprises of the implementation and results details, while comprehensive discussion on system performance and accuracy is presented in Section 5. Finally we conclude our findings in Section 6 and talk about future directions.

2. Related Work

Change is the only constant. Changes in ontologies are always expected to accommodate new knowledge of the domain. These changes are mostly due to the uncontrolled, decentralized, and complex nature of the Semantic Web, which makes ontology change management a complicated and multifaceted task. As mentioned above, ontology evolves as conceptualization of the domain changes. These changes are very important so should be maintained properly for future use.

Protégé provides change management facilities, but all the changes plus the command history are stored temporarily in a file as a script for undo/redo operations [7]. When the project is closed then all these changes and the command history entries are lost.

Y. David Liang in [8] presented a concept of Log Ontology. The author logged multiple ontology changes in Log Ontology II. Then these logged changes are used for the purpose of query reformulation over the evolved ontology to answer user queries transparently. *CRM*[1] ontology is used for system evaluation.

The system in [9] provides two basic facilities: *Change Tracer:* Changes between two different versions of an ontology are detected using PromptDiff [10] and OntoView [11], and then logged using their own format. *Change History Manager:* It keeps user informed about logged changes and their effects on the ontology. The *ChangesTab* [13] of Protégé capture different changes but it does not provide the facility for ontology recovery.

All the above systems provides the facility to log the changes temporarily [7] and persistently [8, 9 and 13]. But none of these systems provides the facility for proper undo and redo operations. They do not use the log for ontology recovery, reverting ontology to previous state, and visualization of changes and change effects on ontology. Their log format is also not appropriate for ontology recovery.

3. System Architecture

Here we briefly introduce the *Change Tracer* architecture (see Figure 1), while detail description is given in [6]. For proof of concept, we have developed it as a plug-in for Protégé an ontology editor. The plug-in can also be used in integration with other ontology editing tools provided they support the hooks we have implemented.

Change Listener: Change listener is a module that actively listen all types of changes happening to ontology model opened in protégé. Whenever a change is triggered, it collects complete information about the change.

For listening changes, we have implemented the action listener interfaces provided in PROTÉGÉ and PROTÉGÉ-OWL API's (see Table 1). For every change their respective listener interface action is triggered and change listener capture information of that change, available in event object of method implemented in that interface.

Change Logger: In this module, all the changes, captured in previous module, are logged using CHL with conformance to CHO.

We claim that we handle all the changes at atomic level; no matter it is atomic change (e.g. deleting a single concept) or complex change (e.g. deleting a sub tree). Atomic change is simple, let's consider the complex change. Consider the CHO given in Figure 2; if we delete *Change_Agent* class then it is a

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

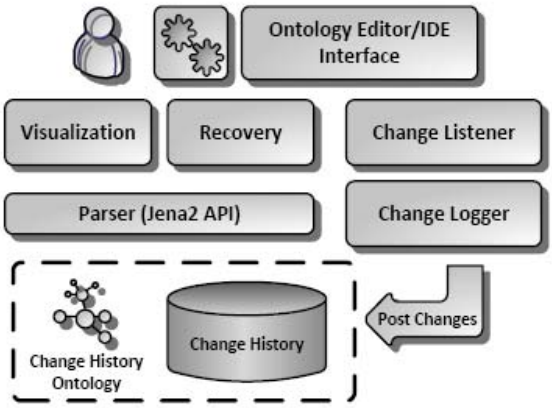

Figure 1. System architecture for Change Tracer

complex change, because it will also result in deletion of its subclasses. Protégé internally provide us the facility that when such a deletion occur, then first its leaf level classes are deleted one by one and then classes a level above leaf level are deleted and so on going to the top. So every deletion triggers its own event, we capture these changes atomically and then its information is logged.

When this module is activated then it first initializes the ontology model from CHO and logs all the changes in CHL using CHO representation.

Change History Log: To recover database from failure, several different techniques [12] are used, to name few: logging, checkpointing, shadowing, and differential tables. Among these, logging technique is most practical and most suited for recovery. We have adopted the logging technique of database recovery for ontology recovery. We log each and every single event related to ontology change, which later on help us in cases like abnormal shutdown, and closing

Table 1: List of Change Listeners Implemented in the Change Tracer Plug-In.

Change Listener	Description
ProjectListener	It listens all the project related events: like saving, closing, form changed, and runtime class widget created.
KnowledgeBaseListener	Helps in listening changes related to the model. It overlaps in its provided methods with all the listeners listed below.
ClsListener	Helps in capturing the class, sub-class, and super-class level changes.
SlotListener	Helps in capturing the slot, sub-slot, and super-slot level changes.
FacetListener	It helps in capturing the changes, such as restrictions, on frames.
InstanceListener	It helps in capturing changes related instances and individuals.

model in Protégé without saving or discarding changes, to undo/redo changes and get ontology to some consistent state.

Change History Log [5] is a repository that keeps track of all the changes made to ontology. It is also required for reversibility purpose when an ontology engineer wants to roll-back or roll-forward some of the changes then this log is accessed and changes are simply reverted. The log uses Jena based triple store and change description is provided by CHO [5] given in Figure 2, to preserve changes for later use.

Parser: Parser job is to: 1) Parses CHL for all the *Change_Set(s)* that corresponds to the open model in Protégé on user request. 2) It also produce the reverse changes of the stored ones, because, user might require to recover pervious state of ontology, then all the *Range_Addition* instances will be converted *Range_Deletion* as shown in Figure 3. The sequence of applying changes back is also in backward order,

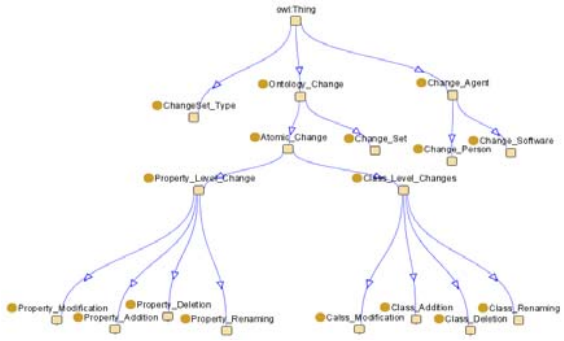

Figure 2. Change History Ontology (CHO)

```
log:Range_Addition_Instance_1224702072640                    A
a                       cho:Range_Addition ;
cho:hasChangedTarget doc:hasAuthor ;
cho:hasPropertyType     owl:ObjectProperty ;
cho:hasRange            doc:Author ;
cho:hasTimeStamp        1224702072640 ;
cho:isPartOf            log:Change_Set_Instance_2474557 .
```

```
log:Range_Deletion_Instance_1224702072640                    B
a                       cho:Range_Deletion ;
cho:hasChangedTarget doc:hasAuthor ;
cho:hasRange            doc:Author ;
cho:hasPropertyType     owl:ObjectProperty ;
cho:hasTimeStamp        1224702072640 ;
cho:isPartOf            log:Change_Set_Instance_2474557 .
```

Figure 3: Using N3 notations, (A) shows the change log entry for Range_Addition change, while (B) shows the parsed listing of (A) for reverting the change.

i.e. changes in a *Change_Set* applied at the end will be reverted first, then second last changes and so on. Then these reverse changes are given to the recovery module which implements these changes.

Recovery: Recovery module is responsible for implementing the applied changes on model opened in Protégé, in forward and reverse manner, based on user request.

When user request to undo/redo any changes or request for recovering previous consistent state of ontology, then this module is activated. For any of the above requests, this module makes request to parser module to retrieve the required *Change_Set* entry and all its corresponding changes. Parser then make reverse changes of all those. When parser returns the reverse changes of the corresponding logged *Change_Set*, then recovery module implements it on the opened model. Algorithm 1 is the roll-back algorithm to recover the model previous state. To extract all the changes corresponding to a specific *Change_Set* instance and their details, we have tested SPARQL query given in Figure 4. It provides all the changes and then these changes are reverted one by one for roll-back and forward based on user request.

Algorithm 1 rollBack (Process ChangeSet): This algorithm assumes a pre-defined function, **TimeIndexedSort** for sorting member entries of the *ChangeSet* based on their timestamp.
Input: An ontology *O*.
Input: An instance of *ChangeSet*, $S_\Delta \in$ *ChangeSet*, which lists changes made in the ontology *O*.
Output: The previous version O' of the ontology *O* after reverting the changes mentioned in S_Δ.

```
1.  /* Sort member entries of the ChangeSet in descending order
    of their timestamp */
2.  TimeIndexedSort(S_Δ, 'DESC')
3.  foreach C_Δ ∈ S_Δ do
4.    /* Process class or role addition */
5.    If C_Δ : OntologyChange ⊓ ∃ changeType.Create then
6.      /* Remove the added resource(s), target of the change */
7.      O ← O – {x | < C_Δ, x > changeTarget}
8.    else
9.     /* Process class or role deletion */
10.     If C_Δ : OntologyChange ⊓ ∃ changeType.Delete then
11.       O ← O ⊓ {x | < C_Δ, x > changeTarget}
12.     else
13.      /* Process class or role modification */
14. ……
15. /* Implementation of this algorithm consist of a number of
    other if-then statements to check type and to process it
    accordingly, such as for annotation */
16.   endif
17. End
```

Algorithm 1, Roll-back ontology changes

Visualization: Visualization module is responsible for visualizing the ontology, ontology changes, and the change effects on ontology. The visualization is in graph like structure rather than tree like structure, because the ontology with class and subclass hierarchy can also have associative relationships with other classes.

Jobs of this module are: 1) A user can request for visualization of the changes in CHL, then these changes are simply parsed and passed to visualization module by the parser without performing any revert operations. On receipt, these changes are visualized. 2) The visualization module also visualizes the current loaded model in Protégé to the user on request. 3) If user wants to visualize the history of ontology evolution process, then visualization module will first request the recovery module to revert the current state of ontology to its previous state with the help of corresponding *Change_Set* changes. The recovery module request parser to extract the required *Change_Set* with complete changes and produce its reverse changes. The result of the parser is implemented by the recovery module, which revert ontology to its previous state and return that state to visualization module, which then visualizes it. The same way if user keeps on requesting, then the same steps are followed. If, at some previous state of ontology, user wants to shift to its next state, rather than previous, then the steps will be same but the parser will not generate the reverse changes, as by implementing the next logged *Change_Set* changes will get ontology to next state.

4. Implementation and Results

We envisioned our proposed framework as an enabling component for the ontology editors. In itself it doesn't provide ontology editing services. The framework architecture is designed to be implemented as a plug-in for different ontology editors provided they support the hooks we have implemented. Different individual components in the framework have their own tasks, related to change history management. Change Logger component, for instance is responsible to preserve the changes and the recovery component on top of all other components should provide ontology recovery services.

To validate the working of the proposed framework, we have developed a *TabWidget* plug-in, *Change Tracer* Tab, for Protégé ontology editor; where detail procedure for plug-in development is available in [14]. The details of all the five main modules and their implementations are available in [6]. Here we provide the evaluation details of the system.

```
SELECT ?change ?changedTarget ?isSubClassOf ?isSubPtyOf ?hasPtyType ?oldName
 ?changedName ?hasDomain ?hasRange  . . . . .?timeStamp
WHERE
{
                ?change   docLog:isPartOf                changeSetInstance .
OPTIONAL  {?change   docLog:hasChangedTarget      ?changedTarget} .
OPTIONAL  {?change   docLog:isSubClassOf          ?isSubClassOf} .
OPTIONAL  {?change   docLog:isSubPropertyOf       ?isSubPtyOf} .
OPTIONAL  {?change   docLog:hasPropertyType       ?hasPtyType} .
OPTIONAL  {?change   docLog:hasOldName            ?oldName} .
OPTIONAL  {?change   docLog:hasChangedName        ?changedName} .
OPTIONAL  {?change   docLog:hasDomain             ?hasDomain} .
OPTIONAL  {?change   docLog:hasRange              ?hasRange} .
                .
                .
                ?change   docLog:hasTimeStamp          ?timeStamp
}
   ORDER BY DESC(?timeStamp)
```

Figure 4, SPARQL query for extracting changes with their details for the given *Change_Set* instance

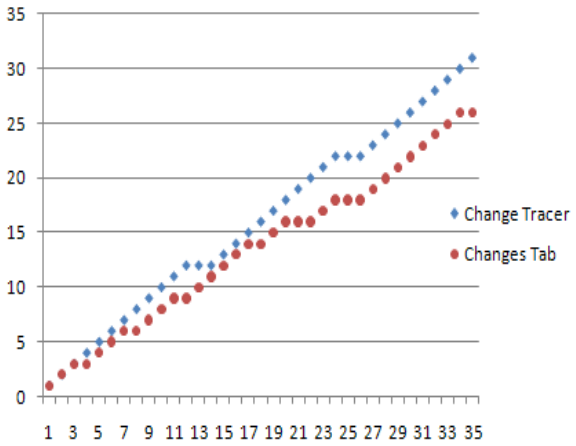

Figure 5, Comparison of Change Tracer against Changes Tab of Protégé

For the development and testing of the plug-in, *Documentation* ontology is used. First of all the results of the plug-in for change capturing are provided using the *Documentation* ontology. The plug-in is also compared with *ChangesTab* of protégé for its change capturing capability.

Change Capturing: There is no such system available (that claim all the features our plug-in provides) to compare the plug-in with. But a Protégé plug-in (i.e. *ChangesTab*) is available that do provide the change capturing facility.

To evaluate the change capturing capability of the developed plug-in, we compared it with the *ChangesTab* of Protégé. Both the plug-ins *(i.e. ChangesTab* and *Change Tracer*) were enabled in Protégé and changes were made to ontology (*Documentation Ontology*) in Protégé. **35** different changes were made to the *Documentation* ontology covering all the four different categories (i.e. Change in Hierarchy, Change in Class, Change in Property, and Other Changes). Out of these **35** changes, *ChangesTab* of Protégé was able to capture **26** changes while our plug-in i.e. *Change Tracer* captured **31** changes. The graph representing these results is given in Figure 5; where y-axis represents the number of changes captured and x-axis represents the number of changes made.

5. Discussions

The aim of this discussion is to validate whether the proposed algorithm for ontology recovery is correct and can scale up to complex ontologies. Validation and verification of the outcome of the recovery process is an essential and critical aspect. There has to be a mechanism to prove the hypothesis that the output ontology after applying the recovery process on top of the Change History Ontology (CHO) is correct. In order to quantitatively measure the performance of the recovery algorithm, an evaluation measure has been used which is discussed below.

For the evaluation of the recovery procedure, we have taken two different versions of ontology i.e. $\mathbf{O_{V1}}$ and $\mathbf{O_{V2}}$. Now the changes between the versions i.e. $\mathbf{C_{\Delta}}$ are stored in Change History Log (CHL) using CHO. After identifying and logging the changes between the two versions, we come up with equation

Table 2. Roll Back and Roll Forward procedures results

Roll Back				
	Tests	**Correct Results**	**Problem(s)**	**Accuracy**
Initial Attempts:	12	5	Domain Addition, DT Pr. Del. (Range)	41.67
1st Revision:	12	7	Inverse Property	58.34
2nd Revision:	12	12	Nil	**100**
Roll Forward				
	Tests	**Correct Results**	**Problem(s)**	**Accuracy**
Total Attempts:	12	12	Nil	**100**

for the verification of recovery procedure. As our plug-in provides both Roll Back and Roll Forward facilities, so we have separate equations for both of these procedures verification.

Roll Back: To roll back the changes from $\mathbf{O_{V2}}$ to $\mathbf{O_{V1}}$, we simply need to subtract all the changes i.e. $\mathbf{C_{\Delta}}$ from the ontology which are the causes for its $\mathbf{O_{V2}}$ from $\mathbf{O_{V1}}$. Now this subtraction of the changes from $\mathbf{O_{V2}}$ is all made using recovery (roll back) algorithm we have proposed. The equation for verification is as under;

$$\left.\begin{array}{l} \mathbf{O_{Vx} = O_{V2} - C_{\Delta}} \\ \mathbf{difference(O_{V1}, O_{Vx}) = \emptyset} \end{array}\right\} \quad \mathbf{(1)}$$

Now applying the recovery (roll back) process on $\mathbf{O_{V2}}$, then it will return $\mathbf{O_{V1}}$. But we store the recovered version in another temporary version $\mathbf{O_{Vx}}$ and then checking this temporary recovered version against the available version $\mathbf{O_{V1}}$. Here we differ $\mathbf{O_{V1}}$ from the recovered version i.e. $\mathbf{O_{Vx}}$ and if the difference is null (empty) then it means that the recovery process for roll back has given correct result.

Roll Forward: To roll forward the ontology from $\mathbf{O_{V1}}$ to $\mathbf{O_{V2}}$, we simply need to add/apply all the changes i.e. $\mathbf{C_{\Delta}}$ to the ontology, which are the causes for its $\mathbf{O_{V2}}$ from $\mathbf{O_{V1}}$. Now this addition of the changes to $\mathbf{O_{V1}}$ is all made using recovery (roll forward) algorithm we have proposed. The equation for verification of roll forward algorithm is;

$$\left.\begin{array}{l} \mathbf{O_{Vx} = O_{V1} + C_{\Delta}} \\ \mathbf{difference(O_{V2}, O_{Vx}) = \emptyset} \end{array}\right\} \quad \mathbf{(2)}$$

Applying the recovery (roll forward) process on $\mathbf{O_{V1}}$, it will return $\mathbf{O_{V2}}$. But here we also store the recovered version in another temporary version $\mathbf{O_{Vx}}$ and then checking this temporary recovered version against the available original version $\mathbf{O_{V2}}$. Then we differ $\mathbf{O_{V2}}$ from the recovered version i.e. $\mathbf{O_{Vx}}$ and if the difference is null (empty) then it means that the recovery process for roll forward has given correct result.

For getting the difference between two ontology models we have used the *difference()* method of *Model* class from *Jena* API. We have also checked both these in Protégé using *PromptTab*. Using the *Documentation* ontology, we have tested the roll back and roll forward algorithms and got very good results. The details of those results are given in Table 2, while their descriptions are given below.

For Roll Back, we tested the plug-in for **12** times and we obtained **5** correct results. The problems are; (1) when a *Domain_Addition* entry is rolled backed then it is reverted as *Domain_Deletion***.** So the algorithm actually has deleted the domain of some property, but Protégé internally assigns *owl:Thing* as domain to all those properties which do not have any domain. (2) When datatype property is deleted then range of that property was not captured properly. Because of these two problems we had very low accuracy level of our plug-in for roll back. We solved these problems and then tested the plug-in for **12** more times and obtained **7** correct results. This time we had only one problem i.e. when a property is made as inverse property then information about the other property to which this property is made inverse to, is missing. We corrected all these problems and conducted **12** more experiments. This time we got **12** correct results and have no problems.

Roll Forward has been implemented after we have completed the implementation of Roll Back and removed all the problems which we faced during roll back. That's why, out of **12** roll forward experiments, we obtained **12** correct results with **100%** accuracy.

The changes capturing ability of the developed plug-in has been compared with the *ChangesTab* of

Table 3, Number and types of changes among different versions of OMV ontology

Ontology Versions	OMV.owl & OMV-0.7.owl	OMV-0.7.owl & OMV-0.91.owl
Total Changes	**38**	**189**
Change in Hierarchy	18	71
Change in Classes	6	34
Change in Properties	25	123

Protégé and the results show that the plug-in have better accuracy than the *ChangesTab*. The recovery algorithms (i.e. *Roll Back* and *Roll Forward*) are tested on *Documentation* ontology and results of high accuracy are achieved.

Evaluation Using OMV: OMV is Ontology Metadata Vocabulary and is used by the community for better understanding of the ontologies for the purpose to properly share and exchange the information among organizations.

To achieve this goal, this standard is set and agreed by the community for sharing and reuse of ontologies. OMV actually provides common set of terms and definitions describing ontologies, so called ontology metadata vocabulary. OMV have different versions available online containing different set of concepts, properties, and restrictions. We have tested our developed plug-in on three different versions of OMV. The OMV[2] versions we have used for the experimentation are omv-0.6.owl, omv-0.7.owl, and omv-0.91.owl.

Table 3 shows complete details about the types and number of changes among different versions. These changes are captured and stored in CHL with the help of *Change Tracer*. Using these logged changes we applied the roll back and forward procedure given in equations 1 and 2, which resulted in recovered versions. We have checked all the recovered versions with the original version and they all were correct.

6. Conclusions and Future Work

Ontology change management is relatively new area of research. Most of the existing research work is based on the ideas from other related fields such as database recovery, ontology merging, and integration.

In this paper we have shown the experimental results for the system and the semantic structure for logging ontology changes. As clear from all the discussion above that the backbone of the system is the *Change History Ontology*, which acts as a glue to bind different components in the framework. The changes logged in *Change History Log* guarantee effective recovery (roll back and roll forward).

The framework for traceability of ontology changes is validated by developing and implementing a plug-in for the Protégé editor. We are planning to extend the framework by using Change History Log to understand the semantic of different changes on the existing constructs in ontology. Reestablishment of ontology mappings based on the stored changes is also in pipeline.

7. Acknowledgment

This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement)" (IITA-2009-(C1090-0902-0002)) and was supported by the IT R&D program of MKE/KEIT, [10032105, Development of Realistic Multiverse Game Engine Technology].

This work also was supported by the Brain Korea 21 projects and Korea Science & Engineering Foundation (KOSEF) grant funded by the Korea government(MOST) (No. 2008-1342).

8. References

[1] G. Flouris, D. Plexousakis, and 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.

[2] 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.

[3] 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.

[4] 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.

[5] A. M. Khattak, K. Latif, S. Khan, N. Ahmed, "Managing Change History in Web Ontologies," Semantics, Knowledge and Grid, International Conference

[2] http://ontoware.org/frs/?group_id=39

on, pp. 347-350, 2008 Fourth International Conference on Semantics, Knowledge and Grid, 2008.
[6] A. M. Khattak, K. Latif, S. Khan, and N. Ahmed, "Ontology Recovery and Visualization," Next Generation Web Services Practices, International Conference on, pp. 90-96, 2008 4th International Conference on Next Generation Web Services Practices, 2008.
[7] Y. D. Liang, H. Alani, and N. Shabdolt. "Ontology Change Management in Protégé". In: AKT DTA Colloquium, Milton Keynes, United Kingdom. 2005.
[8] Y. D. Liang, "Enabling Active Ontology Change Management within Semantic Web-based Applications". Mini PhD Thesis, University of Southampton, 2006.
[9] D. Rogozan, and G. Paquette. "Managing Ontology Changes on the Semantic Web". IEEE/WIC/ACM International Conference on Web Intelligence. 2005.
[10] N. Noy, S. Kunnatur, M. Klein and M. Musen, "Tracking changes during ontology evolution", 3rd International Semantic Web Conference, Florida, USA, pp. 259-273, 2003.
[11] 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.
[12] R. Elmasri, and S. B. Navathe, "Fundamentals of Database Systems (4th Edition)", Publisher: Addison Wesley, 2003.
[13] W. Liu, T. Tudorache and T. Redmond, "Changes Tab in Protégé", http://protegewiki.stanford.edu/index.php/Changes_Tab
[14] M. Musen, N. Noy and Team, "Protégé Developer Documentation", http://protege.stanford.edu/doc/dev.html