Ontology Evolution and Challenges

A. M. Khattak[a]*, Z. Pervez[a], A. M. Khan[b], K. Latif[c], S. Y. Lee[a],

*Corresponding author: asad.masood@oslab.khu.ac.kr,
Tel: +82-31-201-2514, Fax: +82-31-202-2520
[a] Department of Computer Engineering, Kyung Hee University, Korea.
[b] Computer Engineering Department, Ajou University, Korea
[c] School of Electrical Engineering and Computer Science, NUST, Pakistan.

Abstract

Information semantics and semantic interoperability among applications, systems, system agents, and web services are mostly based on ontology. Its increase usage in Information Systems and Knowledge Sharing Systems raises the importance of ontology maintenance. Ontology change management is a collaborative process that incorporates areas like ontology engineering, evolution versioning, merging, integration, and maintenance. Changes are made to the body of knowledge as experts develop a better understanding of the domain. As a result, the body of knowledge evolves from one state to another. Preserving consistency, while accommodating new changes, is a crucial task that needs special attention. This paper aims at providing a comprehensive review on key approaches followed in the field of ontology evolution. The analysis reveals that different individual components have been developed but a complete integrated system for automated ontology evolution is not available yet. This paper also introduces some unfolded challenges in the field of ontology evolution, which must be tackled to complete the process automatically. Moreover, the new changes (evolved ontology) could affect the dependent data, applications, systems, and services. Therefore, this paper also discusses in detail why special attention must be paid to minimize the after effects of ontology evolution and proposes some possible solutions to achieve this goal.

Keywords

Knowledge Management Applications, Ontology Change Management, Ontology Evolution

Ontology Evolution and Challenges

1.0 Introduction

The fundamental aspect of information exchange among applications, systems, system agents, and web services is the development of a consistent and comprehensive model for representing the domain knowledge. It is essential for: sharing knowledge of research outcomes, sharing information among independent organizations [5], exchange of information among clinics [17], and among heterogeneous systems [6]. To make this possible, we need to carefully model the domain knowledge while preserving its semantics [20 and 55].

Ontology provides formal structure (model) with semantics about how an expert perceives the domain of interest with its real meanings. Philosophical ontology is *the science of what is, of the kinds and structures of objects, properties, events, processes, and relations in every area of reality.* In computer science, *ontology is defined as formal and explicit specifications of a shared conceptualization of a domain of discourse* and is the main driving force behind Semantic Web vision [4]. Ontologies are complex in nature and often large structured. Their development and maintenance incorporates related research areas like: engineering, evolution, versioning, merging, and integration where these research areas are fundamentally different [12].

Different convergence technologies like: Semantic Web Services [36, 45], Context-aware Search Engines [26], Software Agents [9], Semantic Grid [46], and Cloud Computing [7] use ontology for their customized needs [2]. Systems using context-aware information (information modelled using ontology) offer opportunities for applications, services, application developers, and end users by gathering context information. The modelled information facilitates in adapting systems behaviour according to application and end user customized needs. Especially in combination with mobile devices, this modelled information is of high importance that increases usability of information and applications on top of the modelled information tremendously [2]. Mostly, systems, services, and technology developed and used by independent organizations are autonomous in nature and behave according to organization's needs. The convergence of diverse systems, services, and technologies in use is based on material unity (information modeling and exchange) at every level and technology integration for that level of unity [33, 46]. Currently in use and most appreciate approach for information modeling, mediation, and integration is use

of ontology in every aspect of data level, system level, service level, and technology level integration and interoperability [2, 15, 20, and 33].

Thus the use of ontology is increasing in Information Systems and Knowledge Sharing Systems, which in response increases the significance of ontology maintenance [12, 13, and 55]. Ontology change results in evolution of ontology where ontology evolution is the change in expert's perception about the domain in view [40]. The evolution process deals with the growth of ontology. 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* [18]. 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 [49]. Ontology change management thus deals with the problem of deciding the modifications that should be performed in ontology, implementing these modifications, and managing their effects on dependent data structures, ontologies, services, applications, and agents [12, 31, and 55].

Ontology evolves from one consistent state to another [19] and to accomplish the evolution process several different sub-tasks are performed in a sequence, i.e., *Capture change, Change representation, Semantics of change, Change implementing and verification, and Change propagation* [12, 13, 15, 23, and 49]. Research on ontology evolution is being carried out by different researcher's groups, and their approaches overlap with each other [12, 13, 18, 23, 31, 40, and 49]. These approaches do have some pragmatic advantages and disadvantages. The current ontology evolution techniques have several hidden weaknesses which are still needed to be unfolded for the purpose of automatic ontology evolution and minimizing its after effects on ontology, applications, and services. One major weakness is that the specification of new changes due to change in domain knowledge, resolving inconsistencies because of new changes (selecting deduced changes from available alternatives), and also undo and redo in case we want to recover the ontology are all done manually [24]. In order to automate the process of ontology evolution, we need to automate all the above mentioned tasks. This automation is important because human intervention is time consuming and error prone. In addition to these issues, the process of evolution also brings consequent effects on dependent applications and services using the evolving ontology, which must be minimized [15, 25, and 29].

The goal of this research is to provide a comprehensive review of the approaches employed by different research groups for ontology evolution. The paper discusses in detail the main features of these approaches, and their contributions. Their limitations are also highlighted using summary tables and are critically analysed. Furthermore, the paper discusses some open challenges that need to be addressed in order to completely automate the process of ontology evolution. We have also discovered and highlighted consequent issues/effects of ontology evolution on dependent ontology, applications, and services and have also suggested possible solutions to minimize these effects. .

This paper is arranged as follows: Section 2 provides a case study on ontology change and the effects of ontology change on a service using ontology. Section 3 briefly discusses different types of ontology changes (reasons for ontology evolution) and change management activities to cope with these changes. Section 4 presents ontology evolution approaches proposed by different researchers with their contributions and limitations. In Section 5, we present the challenges that are still needed to be tackled for complete automation of evolution procedure and minimize the after effects of ontology evolution on the dependent data, systems, and technologies. Finally, we conclude our discussion in Section 6.

2.0 Ontology Change a Case Study

Ontology is being used by different convergent technologies for their customized needs, such as Context-aware Search Engines [26], Software Agents [9], Semantic Grid [46], and Cloud Computing [7]. In this section, we mainly focus on Semantic Web Services [36, 43, and 45] (as a road map for later sections) and their use of ontology for the completion of different tasks. Current web is the migration of traditional web from collection of web pages to collection and integration of services that can interoperate with one another. Interoperability is a collaborative and multifaceted task to overcome the problems of incompatibilities among organizations, structures, data, architecture, services, and business rules [30 and 43].

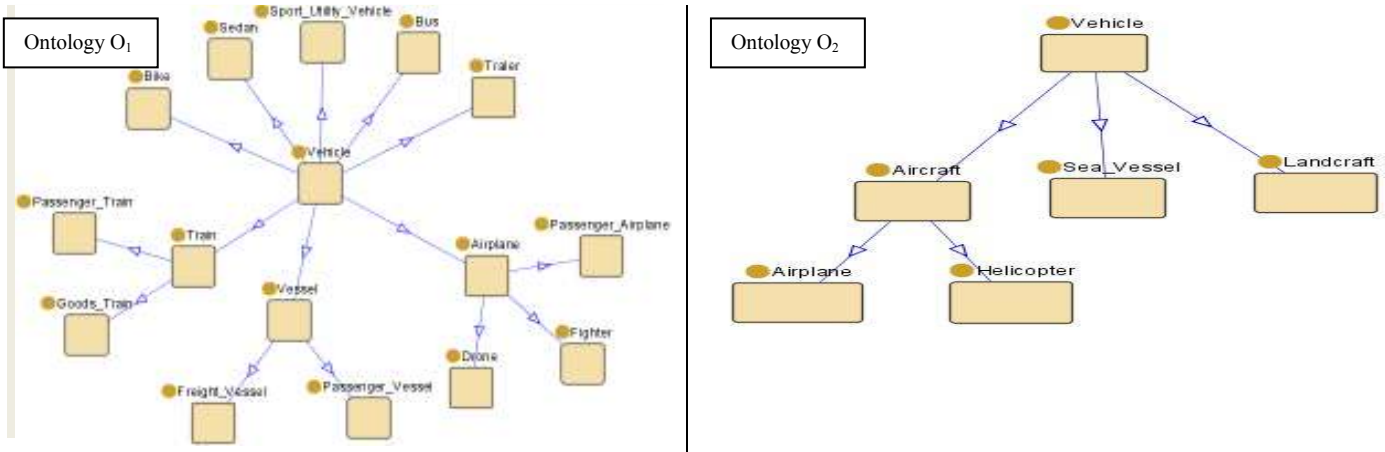

Figure 1, Vehicle ontologies, Ontology O_1 is used by one Web Service and Ontology O_2 is used by another Web Service.

In this section we only highlight the use of ontology for web services in order to set the stage for the upcoming sections that would focus on issues that a web service might face due to ontology evolution. The *Vehicle* Ontology O_2, given in Figure 1, is an extended version of the ontology used in [43]. Consider two web services that provide services about *Vehicle*. These services use the two ontologies given in Figure 1 respectively, to fulfil client's diverse requests. Now consider a scenario in which a client requests for *Train* related information from the service that is using ontology O_2, but the requested information is not available with the contacted web service. This web service has a collaboration established with another web service that is using ontology O_1, so it will route the client's request to the second web service where client's needs are fulfilled. For this routing of request, proper mapping of information is required from both sides [10, 16, 35, and 43] so that semantic interoperability could be achieved. However, consider that the same ontologies on both sides are used by different stakeholders and they make the changes in a centralized instance of ontology. The change happened to any side of the collaborating ontologies will make the established mappings unreliable and there will be no more information sharing among the two services (see Figure 2). To evolve the mappings and continue the sharing of information, proper maintenance of ontology changes and their reflection is necessary [12, 15, and 27].

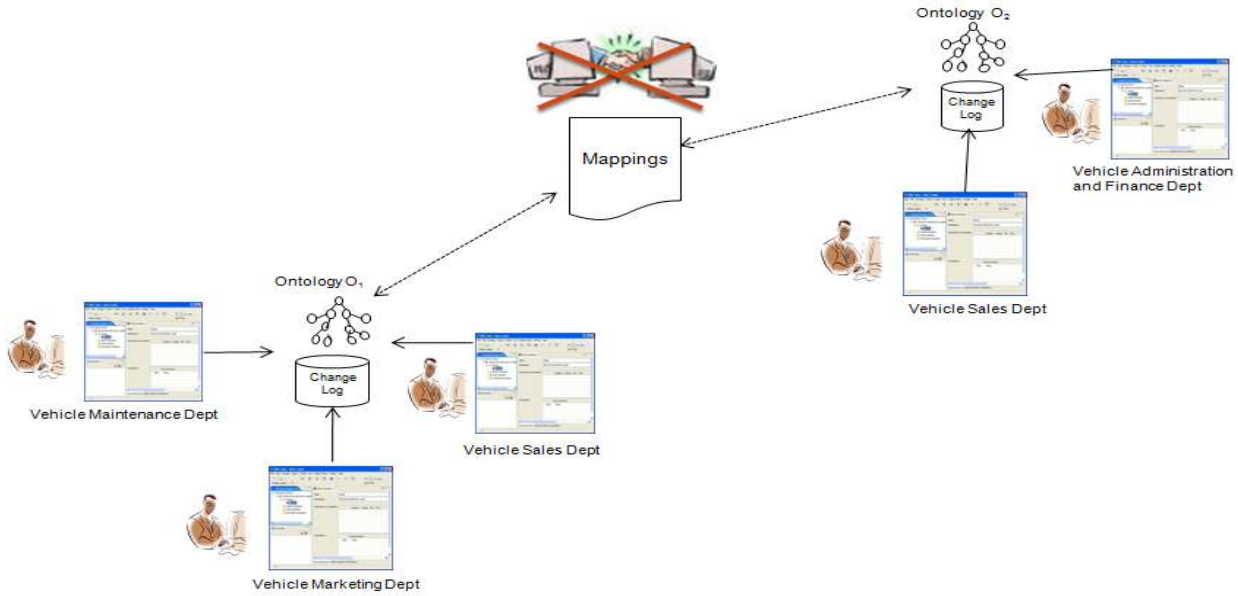

Figure 2, Vehicle ontologies, Ontology O_1 and Ontology O_2 used for information sharing scenarios. Both the ontologies are sharing information based on the established mappings. However, changes initiated from any department will make the mappings unreliable which will stop the sharing of information between the two ontologies.

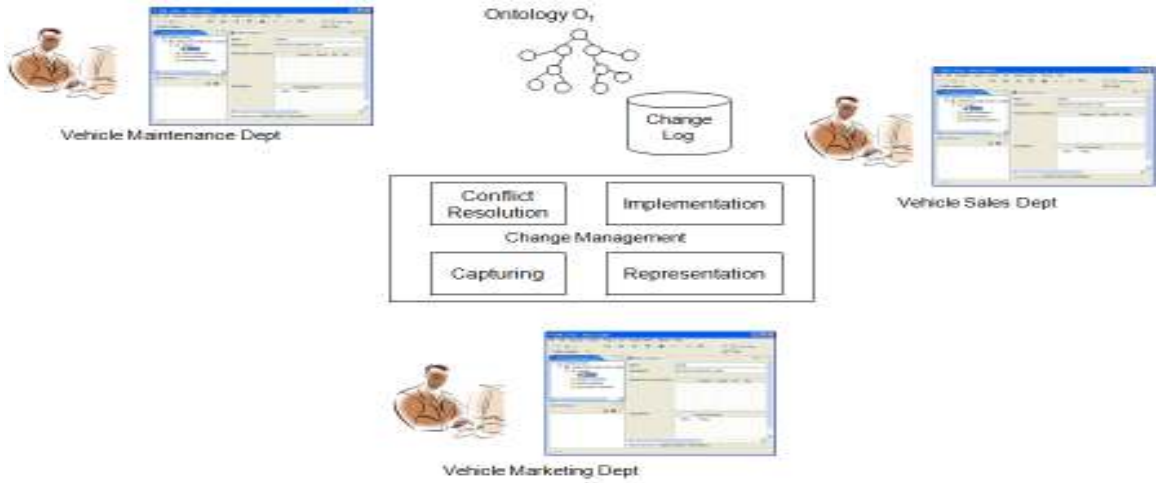

Figure 3, Changes in Vehicle ontology from different stake holders and their management using the ontology change management module for consistent evolution of ontology from one state to another.

To facilitate the dynamic mappings in this case, proper management of ontology changes is required. This gives importance to ontology change management to support smooth evolution of ontology from one consistent state to another [12]. Consider the scenario given in Figure 3 where the same *Vehicle* ontology (centralized in this case) is continuously subject to change because of the participating stakeholders. Now the changes need to be reflected on the final state of ontology without any conflicts for inconsistencies. This task is carried out by ontology change management module by incorporating the evolution procedure. The requested changes are semantically

represented. Possible conflicts are resolved and the changes are implemented on the ontology that evolves to another consistent state.

3.0 Ontology Change Management Activities

As explained in [12] ontology change management deals with the problem of deciding the modifications to be performed in ontology in response to a certain need for change. The ontology change management is a mechanism that keeps the changing ontology consistent, keeping it updated with all the required changes reflected in it. Different changes have different effects on overall ontology and most of these changes are discussed in [8 and 31]. Ontology change management deals with four main activities which are related and in some cases overlap but are fundamentally different activities: *Ontology Evolution, Ontology Versioning, Ontology Merging, and Ontology Integration.*

- *Ontology evolution* is the process of modifying ontology in response to a certain change in the domain or its conceptualization [8, 12, 13, 31, and 49].
- *Ontology versioning* is the ability to handle an evolving ontology by creating and managing its different versions [12, 13, 52, and 53].
- *Ontology integration* is the process of composing an ontology on a particular subject from information found in two or more ontologies covering multiple/diverse domains [12, 13, and 51].
- *Ontology merging* is the process of composing an ontology from information found in two or more ontologies covering highly overlapping or identical domains [12, 13, and 50].

A number of changes, ranging from concepts to properties, could affect the ontology when it is requested to be reflected in the existing ontology. Most of these changes are discussed in greater length in [31]. Here we will briefly highlight some of the aspects that will initiate a change when requested for accommodation in the ontology.

- *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 different 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 *Luxury_Vehicle* could be a separate subclass of *Vehicle* or could be modeled as property of the concept *Vehicle*.
- *Concept with Changed Hierarchy:* Different modeling approaches may fix the same concept in different hierarchical locations in two different ontologies.

A single change in ontology can be of both simple and complex nature depending upon resources that are affected with it which consequently affects the depending service. Understanding change types (i.e., simple and complex) is necessary to correctly handle explicit and implicit change requirements [18], and consequently understand the effects of these changes on ontology and ontology based web services. Renaming a class or a property could be regarded as simple changes, whereas merging two hierarchies with all their constraints could be termed as a complex change. Keeping these simple and complex changes in view, ontology changes are arranged in different forms listed below, but these classifications of changes do overlap [31]. 1) Changes at class level correspond to those changes that are directly related to classes. For example adding, deleting, updating, renaming, and merging different classes. 2) Changes at slot level refer to changes which are related to slots, such as adding, deleting, updating, and renaming different slots. Other examples of slot level changes include setting domain/range of slots, setting the slot as symmetric, functional, and inverse. 3) Change in hierarchy of the ontology means modifying the structure of the ontology. These changes include adding, deleting, moving, and merging different classes and subclasses, slots and sub-slots in an ontology. So these changes depend on class-level and slot-level changes 4) Change at instance level is a kind of change that occurs when the instances are added, deleted, and modified. The modification at instance level change can be updating its attributes values. Some other examples include changes in property characteristics, equality or inequality, restricted cardinality, and union or intersection.

Different changes may introduce different issues in ontology as well as in dependent services. These issues and their possible solutions are discussed later. For instance a simple change, such as deletion of a class from a hierarchy, could result in numerous complex changes. Let's consider the Ontology O_2, given in Figure 1. Suppose that a simple change of class (i.e., *Sea_Vessel*) deletion occurs. (1) This deletion needs some decisions like whether all the instances of *Sea_Vessel* should also be deleted, which is loss of information. If not then how these instances should be maintained in the hierarchy. (2) Suppose that all the sibling classes (i.e., *Aircraft*, *Sea_Vessel*, and *Landcraft*) are disjoint. In that case, instances of *Sea_Vessel* cannot be distributed among the disjoint classes. (3) Consistency of ontology after this change is not guaranteed.

4.0 Ontology Evolution Approaches

Ontology evolution means modifying or upgrading the ontology when there is a certain need for change as *Communities of Practice* concerned with the field of knowledge develops a deeper understanding of the domain. Ontology over time needs to be updated to accommodate new discoveries (changes) in the domain knowledge, user requirements, and to incorporate incremental improvement in the service. The evolution process deals with the growth of the ontology by capturing, and accommodating the new information [8, 12, 13, and 49]. Different ontology editing tools are developed and most of their functionalities are based on the algorithms and approaches discussed later. Table 1 provides a brief investigation of these tools with their contributions and limitations.

Table 1, Brief description of ontology editing tools

System	Contributions	Limitations	Evolution
Protégé [38 and 39]	• Mostly used for ontology creation • Often used for evolution and maintenance • Provides Visualization, Merging, Integration, and Comparison • SparQL queries support	• Weak facility for ontology change management • No facility for ontology recovery • Use third party services for consistency checking of ontology	Manual evolution support
KAON [14]	• Provides ontology editing services like Protégé • Provides good environment for pre-evolution strategy making, generate deduce changes to avoid conflicts • Supports automatic evolution, redo and undo • Provides collaborative editing facility	• Complex system • Slow in response • Needs ontology engineering for conflict resolution	Pre-defined strategy based evolution support
OilED [3]	• Used for ontology engineering • Disallows inconsistency in ontology • Supports semi-automated ontology evolution	• No change logging facility • No facility for ontology recovery • More strict in its operations compared to Protégé	Semi-automatic support
OntoEdit [48]	• Used for ontology editing • Uses more options than KAON for strategy making • Allows collaborative environment for ontology editing	• Provides less operations than KAON • To avoid side effects of conflicts, it involves ontology engineer	Strategy-based evolution support

4.1 Ontology Evolution

The evolution process involves following subtasks (see Figure 4, ontology evolution lifecycle). *Capture Change:* which will capture the required changes to be applied to ontology. *Change Representation:* where all the required changes are represented using formal representational format. *Semantics of Change:* where the effects of the required changes are tested on ontology for its consistency and if required then some deduced changes are also included in the change request to avoid conflicts. All these deduced changes become part of the required changes. This process is mainly related to the area of ontology debugging [12]. *Change Implementation and Verification:* where the complete change request is executed on the ontology and for a validation is performed to see if the requested changes made to ontology or not. In this phase all the applied changes are also logged into a repository (change log) and these logged changes are later used for different purposes. *Change Propagation:* where changes are propagated to all the dependent data, applications, and services.

The evolution in ontology is mainly of two types i.e., *Ontology Population* and *Ontology Enrichment* [25]. *Ontology Population* is when we get new instances for concept(s) that is already present in the ontology. Then this concept(s) is not inserted for the second time. Here only the new instance(s) of this concept(s) is introduced and the ontology is populated. *Ontology Enrichment* is when we get changes in the structure of ontology. For example when we get new concept(s), which is totally new for our ontology or the concept does have some sort of changes from its counter concept(s) in the ontology. Then we enrich our ontology to accommodate the new changes and also populate our ontology for its instance(s). In this section, we briefly discuss ontology evolution approaches with reference to their comparison in terms of their contributions and limitations. At the end, we will critically analyze these approaches that will set the stage for next section on open challenges.

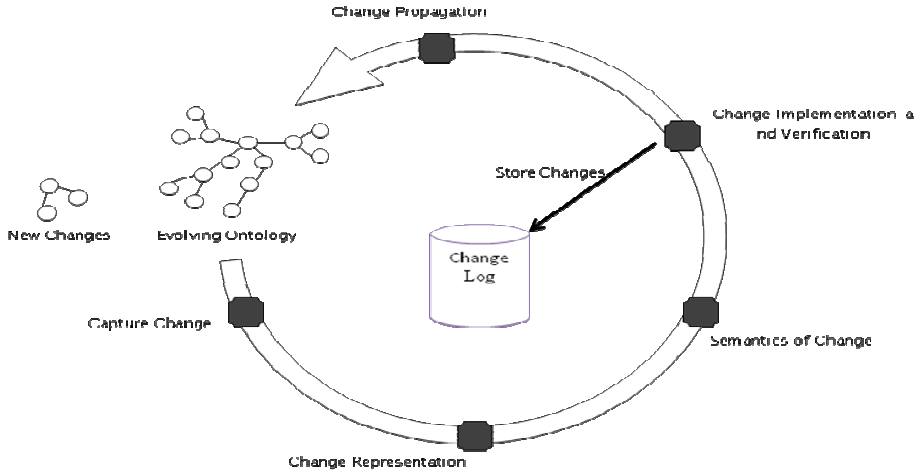

Figure 4, Ontology Evolution Lifecycle, takes source ontology along with new changes and implement the new changes to source ontology.

Initially, L. Stojanovic and B. Motik in [49] talked about the support that different ontology editors provide in order to develop ontology, and also discussed the limitations, complexities, and usability issues of these tools for ontology evolution management. As ontology developed for any domain needs refinements, it must be updated by making the appropriate changes in it. Therefore, methods to cope with the changes that result from ontology evolution are an essential requirement for ontology editors.

Types of change capturing, such as structure driven, data driven, usage driven, and discovery-driven are discussed in [18] and the requirements that an ontology management system (editor) should provide for ontology evolution and propagation of these changes to the dependent data and applications are discussed in [49]. 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 [12, 14, 28, 31, 38, and 56], new changes are specified by ontology engineer, whereas [13, 41, 44, and 47] detect new changes between two different versions of ontologies using PromptDiff (Protégé plug-in), OntoView [21], and H-Match [10] algorithms. The systems discussed in [8, 23, and 56] detect new changes mostly using WordNet and H-Match [10] for finding correspondence among these changes and already existing resources 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) [31], Change Log [47], and Change History Ontology (CHO) [22]. 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

2) ontology engineer resolves these conflicts, but systems like KAON [14] and OntoEdit [48] use a predefined resolution strategy, and Evolva of NeOn Toolkit [56] uses a customized run time strategy for conflict resolution.

After the implementation of the new changes, some systems [14, 23, 31, and 47] provide the facility for change logging for the purpose of undo/redo and ontology recovery (both roll-back and roll-forward) [24]. There is also a need to select an appropriate level of granularity for change item. A very simple level can cause side effects [49] while complex level can make problems during ontology recovery and can pass up the facility for item level recovery [24]. After the validation of implemented changes, they are propagated to the dependent data, applications, artefacts, and services using push-based and pull-based [49] approaches. But most of the evolution systems do not provide 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 entity (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, which excludes change propagation phase from ontology evolution procedure.

TABLE2. Summary of ontology evolution approaches. Last column representsmaturity level of the approach in terms of automation.

Approaches	Change Request	Change Representation	Conflict Resolution	Change Implementation	Change Propagation	Working
L. Stojanovic, et al. [49].	The complete change request is represented in formal representational format. These changes (due to business requirements) are specified by ontology engineer.		Ontology engineer resolves 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, services, and ontologies. Out-of-date instances are simply replaced with the up-to-date instances.	User intervention required for system working
M. Klein, N. Noy, et al. [28, 31, 38]	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 intervention required for system working
T. Gabel, et al. [14] (KAON)	Specified by ontology engineer	Formal representation of changes	Use of predefined strategies for conflict resolution and avoiding side-effects.	Provides interface for user interaction and also logs the changes.	Propagation of changes to dependent artefacts.	Most parts of the system are formalized and working
P. Plessers, et al. [44]	Different versions of ontologies are used in this approach. Changes among different versions are represented formally.		After change implementation, it checks for inconsistencies and if present then it makes the change recovery.	First it implements the change request and then checks for any conflicts.	It does not support change propagation as it works on versions.	User intervention required for system working
D. Oberle, et al. [41, 47]	Changes are detected among two versions by using PromptDiff and OntoView [21], and a complete change request is compiled	Formally represented using their developed semantic structure.	Ontology engineer resolves inconsistencies by introducing deduced changes	With change implementation, all the changes are also logged for undo/redo purpose	It does not support change propagation as it works on versions	User intervention required for system working
P. Plessers, et al. [44]	Use top-down approach (manual) and bottom-up approach (automatic) for change detection	Suggestions for the use of formal representation i.e., using the log representation for changes	Involves ontology engineering for resolving conflicts	Manual implementation of these changes	It does not support change propagation	User intervention required for system working
H. Liu, et al. [32]	Changes are suggested by end user and are assumed to be represented at atomic level.		For all conflicts, the resulting solutions are calculated using DL assertions	Changes are implemented; no log is maintained for this.	Suggestions for consistent propagation are made	User intervention required for system working
S. Castano, et al. [8]	Changes are recognized automatically by analysing domain artefacts. H-Match [10] and WordNet[1] [37] 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 a focus.	Semi automatic
A. M. Khattak, et al. [23]	New changes such as (change in single concept, group of concepts and concepts in a hierarchical structure) are detected automatically using H-Match [10] and WordNet. Change representation is provided by Change History Ontology (CHO) [22].		For conflict resolution KAON API [14] is used with some suggested extensions.	Changes are implemented atomically and after every change implementation, these are logged in CHL [22]. At the end, all changes are validated against the change request.	Change propagation is not handled in this approach.	This approach provide suggestions toward automation of the process
F. Zablith [56]	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.	This approach is a step towards automatic ontology evolution

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

4.2 Discussion

Existing ontology evolution approaches discussed above, to some extent have achieved automatic ontology evolution. However, still they have many issues to consider before announcing an automated system for ontology evolution. Some of the issues, not handled by existing systems that needs to be worked on are discussed briefly in this section.

- The existing systems do not consider new emerging concept(s) (instead they work with manual change requests), which is the first and amongst the most important aspects of developing automated evolution procedure.
- In [49] the authors talked about different requirements that need to be fulfilled in order to achieve ontology evolution properly, such as composing change request, conflict resolution, change implementation, and change propagation; however, they did not provide any tangible results. In [44 and 49], users manually created the requests for changes, whereas the conflicts were manually resolved by experts.
- In [8], author focused on discovery of new change (resource) and afterwards ontology expert inserted the resource at suitable place suggested by the system. Their main achievement was to use matching technique and discover most appropriate position for the new emerging concept in ontology hierarchy. The main concerns in [23 and 56] systems were the best matching resource(s) selection for the newly emerging change (resource).
- Inconsistency resolution is also amongst the most critical problems that needs attention before, during, and after evolution procedure. The consistency is checked for; consistent modelling of new resources in presence of existing resources, 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 [23] the author proposed a training process for different deduced changes. But training a system for different induced changes and their consequent deduced changes is a tough job, and even after proper training the system results may not match user's intentions. For a single conflict there might be many alternative deduced changes, and deciding upon a certain change is another issue in itself [29 and 56].

5.0 Open Challenges

In this section, we explain in detail the challenges that need to be solved for the purpose of achieving automated ontology evolution [25]. Afterwards, we discuss in detail some of the after effects of ontology evolution on the dependent data, applications, systems, system agents, services, and other ontologies. For the challenges discussed in different scenarios, we also suggest possible solutions to overcome these challenges or minimize their effects. In discussion on these challenges, we present the problems at class level (concept) but it is applicable to all including slots, instances, and restrictions. The first two issues arise during evolution whereas the rest is related to the after effects of evolution on agents, services, and ontology.

5.1 Change Detection

The first phase of automatic ontology evolution is to detect new changes in an ontology that is representing changes in domain knowledge. To detect changes among the newly emerging concept(s) (i.e., single concept, group of concepts, and concepts in a hierarchical structure), various resource correspondence, difference, and matching [10 and 41] techniques are applied. This helps in finding out the most appropriate position for the emerging concept in concept hierarchy of the source ontology [12]. First of all, the existence of newly detected resource is checked, and if it does not exist then matching process starts to detect the most relevant concept(s) in the source ontology [8 and 23] where the new concept should be inserted. However, there exist two different problems [25]:

- *Relevance Detection:* It means to calculate the relevance between the new concept and the existing concept(s). The currently used (in practice) algorithms for difference, correspondence, and matching are presented in [10, 16, and 41]. However, their results are still not accurate enough for diverse domains, so using these algorithms is not fully reliable and user intervention is required. See Figure 5, these algorithms [10, 16, and 41] produce high correspondence for the newly emerging concept *Vehicle* having sub-concepts *Light_Vehicle* and *Heavy_Vehicle* against the root concept *Vehicle* of the source ontology (in use of a web service), and produce low correspondence against the *Landcraft* concept. But this correspondence is not correct as the emerging concept *Vehicle* is actually more related to the concept *Landcraft* than the *Vehicle*.

- *Selection among Newly Detected Changes:* It is quite possible that more than one concept is related to the newly detected/emerging concept, so now the problem is that which alternative should be selected. To understand this problem concentrate on Figure 5, where we have emerging concepts *Fighter_Airplane* and *Airliner_Airplane*, and the source ontology is *Vehicle* ontology to which the changes will be applied. The correspondence calculated for these emerging concepts give three alternatives for their insertion in the concept hierarchy: 1) Insert both emerging concepts as sub concepts of *Vehicle* concept. This alternative is reasonable as *Aircraft* is already a sub concept of *Vehicle* concept. 2) A more feasible option is to make both these concepts sub concepts of *Aircraft* concept in the source ontology. 3) Another alternative is to make *Fighter_Airplane* and *Airliner_Airplane* sub concepts of *Airplane* and this is the most reasonable suggestion. An ontology expert knows that the third alternative is the most suitable one, but in automatic evolution procedure the decision is to be made by the system. Proper heuristics should be implemented or the system should be trained for such situations. But it is a tough task to train the system as ontology is very much different in its nature and structure than any other information representation schemes [40]. These issues are still unsolved and need attention.

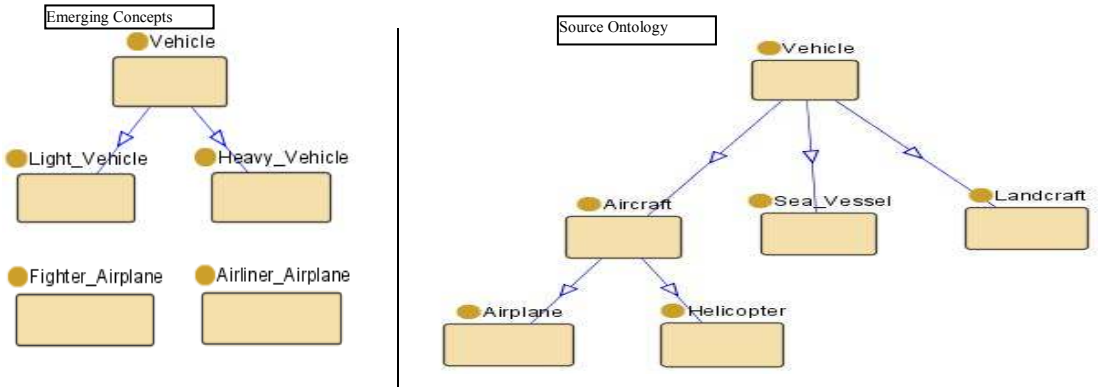

Figure 5, Emerging concepts are the newly detected changes (discoveries) in domain and are the reasons for change in source ontology (i.e., Vehicle Ontology in use of a web service).

5.2 Conflict Analysis

Consistency of ontology after evolution is the most critical concern. Inconsistencies/conflicts may arise in ontology due to changes requested in domain ontology. In order to resolve these conflicts and make ontology consistent, deduced changes are introduced in change request. Introduction of appropriate deduced changes is one of the most highlighted problems in ontology evolution literature. Most of the existing system use expert intervention for

resolving the conflicts [8, 28, 44, and 49]. In KAON [14] and OntoEdit [48], predefined evolution strategies are used to avoid any sort of inconsistency. For example, if there are two alternatives for a concept change: 1) to become a *property* of some concept, and 2) to become a *sub-concept* of some concept in the source ontology (like *Luxury_Vehicle* concept case in Section 3), then the choice of *sub-concept* should be selected which is predefined in the evolution strategy. The problem is, we cannot make predefined strategies for all sorts of conflicts, so ontology engineer is required for conflict resolution [14 and 48]. In [23], we proposed training the system for different types of deduced changes, and then accordingly selecting the alternative (deduce change) that has less impact on ontology. To resolve the conflicts in this way, we need to address two very important things.

- *System Training:* It is very hard to train the system for an exhaustive list of changes (even for a specific domain) and then expecting accurate results. Moreover, the results may also not be acceptable to ontology engineer due to different perception. In addition, there might be cascading conflicts and resolving all these may result in weak response time of the system which in result can make a web service using this ontology to go offline.
- *Impact of Deduced Changes:* The training process may give us different alternatives for deduced changes. In [23], we proposed to select those deduced changes from alternatives having less impact on ontology. However, special attention must be paid in deciding about which aspect of the ontology should be considered to analyze the impact of change for the deduced changes. Moreover, it should also be kept in mind that some changes have larger impact on the structure of ontology but have less impact on the semantics of resources in the ontology. For example (see Figure 5), adding concepts *Light_Vehicle* and *Heavy_Vehicle* as sub-concept of *Landcraft* in source ontology have larger structural impact than semantic impact. In the same way, adding concepts *Light_Vehicle* and *Heavy_Vehicle* as sub-concepts of *Vehicle* in source ontology have less structural impact but more semantic impact. If we make *Sea_Vessel* disjoint with its sibling concepts, then this change also has less structural impact but can have very large impact on semantic of the resources as its effects will also be reflected on the sub-concept(s) of all the disjoint concept(s).

To achieve the automatic evolution procedure for ontology, these problems need to be resolved in order to guarantee a consistent evolution of ontology from one state to another.

5.3 Change Traceability

Corresponding to the CRUD interfaces in databases, there are three categories (excluding read) in the proposed ontology representing the change types: Create, Update, and Delete. There are four categories in the ontology to represent different components of the ontology being subject to change (i.e., *Class*, *Property, Individual, and Ontology* [22]). Based on the above mentioned categories, we derive instances of class *OntologyChange*, represented with the symbol Δ, using the following axioms, for details see [22 and 23].

$R_{\Delta} \equiv \exists$ ChangeTarget.(Class $\sqcup$ Property $\sqcup$ Individual $\sqcup$ Ontology)
$\Delta \equiv R_{\Delta} \sqcap \forall$changeType.(Create $\sqcup$ Update $\sqcup$ Delete) $\sqcap \exists$changeAgent.(Person $\sqcup$ SoftwareAgent) $\sqcap =1$changeReason

Evolution of ontology from one consistent state to another is due to a change in the domain [8, 12, 23, and 49]. These changes and the reasons for these changes need to be preserved for later use. We proposed and use Change History Log (*CHL*) to store all the changes in formal and semantic representational format provided by Change History Ontology [22]. Changes of specific time interval are logged as one *Change_Set* (see Figure 8), and this *Change_Set* changes are the reason for ontology evolution. Managing ontology changes during evolution in *CHL* is also helpful for new users to understand the changes made to ontology. Using entries of *CHL* one can also extract/understand the change in semantics of the changed concept(s). Annotation can also be added with all the changes, such as reason for the change, effects of the change on dependent data, application, services, and other artefacts, which could help in understanding the changes in ontology, data, application, and service behaviour.

5.4 Ontology Recovery

Changes occurring in ontology make the ontology to change from one state to another state. Proper maintenance of these changes is very important to provide the facility of reverting back to the previous consistent state of ontology. These stored changes not only provide the facility for rollback, but are also used for roll-forward operations based on user request. Different ontology editors like KAON [14], Protégé [39], and OntoEdit [48] do provide the facility for undo and redo changes but they do not provide the facility for complete recovery of ontology from one consistent state to another.

We validated our proposed framework as an added component for the ontology editor (i.e., Protégé) [24]. It is designed to be implemented as a plug-in for different ontology editors provided they support the hooks

implemented in our plug-in. The recovery component, on top of all other components, should provide ontology recovery services. For details on recovery procedure and validation of recovery output refer to [24]. The recovery process is still not mature as the proposed method is only valid for structure level recovery of ontology. There is still no system available that can work for both structure and instance level recovery, which will also guarantee smooth operations of a web service using this ontology.

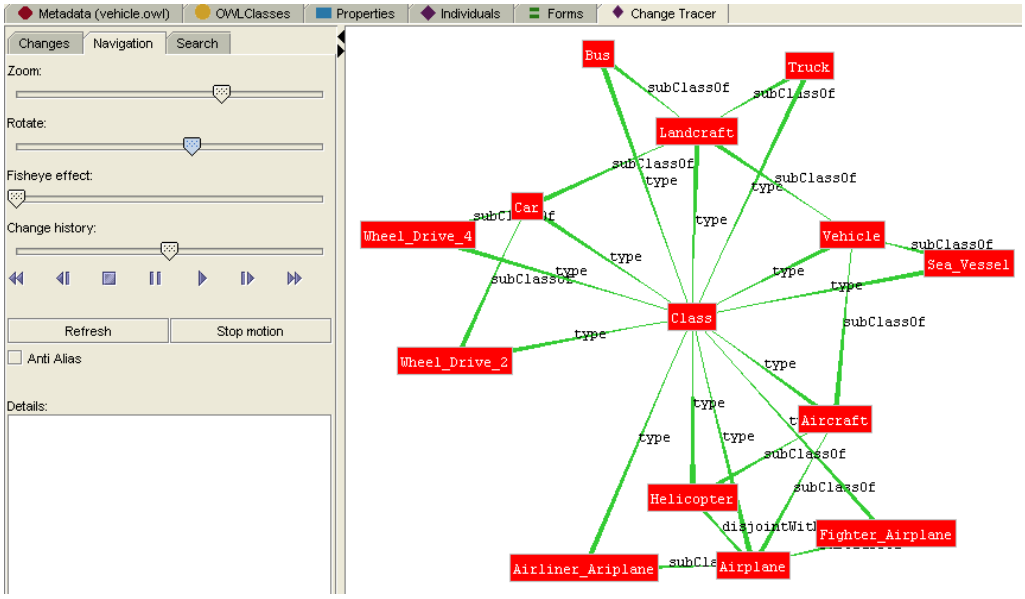

Figure 6, Visualization of Vehicle ontology using Change Tracer [24].

5.5 Change Visualization

Ontology visualization tools and plug-ins are available in abundance. None demonstrates ontology evolution and change visualization. New breed of ontology visualization tools can be implemented using change history log to visualize different ontology states. The ontology changes (logged in CHL) can be used to visualize the change effects on ontology in different states through which it passed before reaching the current state. Such visualization will provide the facility to temporally trace the ontology changes and better understand its evolution behavior. The need is to provide a fully functional system that can visualize the ontology with all its resources and also have the functionality for forward and backward navigation in the history of ontology. This will help in proper understanding of not only the evolution behavior but of ontology as well. Figure 6 provides the visualization of *Vehicle* ontology

using our developed plug-in for Protégé. This visualization feature can give a better view of the information contained in a web service using this ontology. The visualization can help user to visualize ontology at a particular time instant, all changes after that time interval are reverted back and the older version of the ontology is regenerated and visualized.

5.6 Change Prediction

For conflict resolution as well as for future change prediction, the logged changes can be of significant help. The changes logged in CHL [22] are atomic level (simple) changes that do provide lots of open space for change patterns. See Figure 8, where after every new class addition a class renaming change is performed. The same way, after every property addition, property renaming and setting its domain and range is occurring. So these patterns can be extracted and used in conflict resolution and next change prediction that can indirectly help the dependent applications and services. Using these frequent patterns, we can also get a better understanding of development. Figure 7 is an example of mining frequent patterns from some of the logged changes, which are shown in Figure 8.

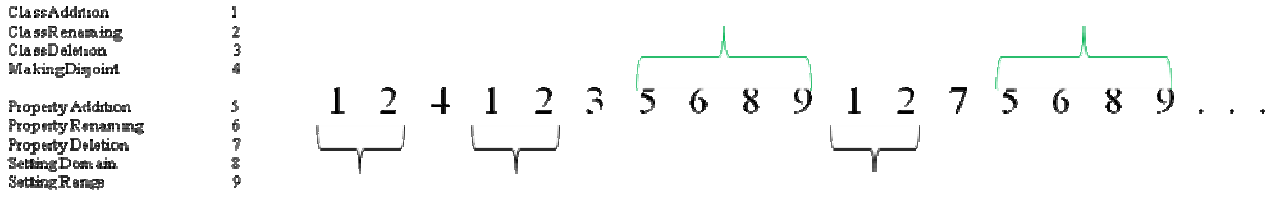

Figure 7, Mining frequent change pattern from logged changes

5.7 Query Reformulation

Query written over one schema does not give correct results when executed over another schema [1], so it needs to be reformulated in order to fulfill the schema requirements. Same is true for ontology, so when ontology evolves from one consistent state to another then the query written over previous state needs to be reformulated to extract the required results from the evolved ontology [34]. The author in [34] 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 [34]). They evaluated the system using two different versions of CRM[2] [11]

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

ontology. The main idea behind this work is to maintain the changes that cause evolution in a repository and later used these changes for query reformulation.

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 versions of ontologies at the same time as it's hard to extract the changes from the log that corresponds to a particular state of ontology. In [22], a *Change History Ontology (CHO)* is presented that logs all the ontology changes in atomic manner and also keeps the changes separate from those that correspond to a different state of

```
log:Change_Set_Instance_2010_08_06_00_05_04
      a       cho:Change_Set ;
      cho:hasChangeAuthor
      log:Change_Person_Instance_2010_08_06_00_03_54 ;
      cho:hasChangeBeginTime "2010-08-18T00:05:04"  ;
      cho:hasChangeReason "New Changes";
      cho:hasOntology "vehicle".

log:Class_Addition_Instance_1282056007953
      a       cho:Class_Addition ;
      cho:hasChangedTarget vehicleClass_1 ;
      cho:hasTimeStamp "1282056007953";
      cho:isPartOf
      log:Change_Set_Instance_2010_08_06_00_05_04;
      cho:isSubClassOf vehicle:Airplane .

log:Class_Renaming_Instance_1282056012843
      a       cho:Class_Renaming ;
      cho:hasChangedName "Drone" ;
      cho:hasOldName "Class_1" ;
      cho:hasTimeStamp "1282056012843" ;
      cho:isPartOf
      log:Change_Set_Instance_2010_08_06_00_05_04;
      cho:isSubClassOf vehicle:Airplane.

log:Class_Addition_Instance_1282056020687
      a       cho:Class_Addition ;
      cho:hasChangedTarget vehicle:Class_2 ;
      cho:hasTimeStamp "1282056020687" ;
      cho:isPartOf
      log:Change_Set_Instance_2010_08_06_00_05_04;
      cho:isSubClassOf vehicle:Airliner_Ariplane .

log:Class_Renaming_Instance_1282056031031
      a       cho:Class_Renaming ;
      cho:hasChangedName "Passenger_Airplane" ;
      cho:hasOldName "Class_2" ;
      cho:hasTimeStamp "1282056031031" ;
      cho:isPartOf
      log:Change_Set_Instance_2010_08_06_00_05_04;
      cho:isSubClassOf vehicle:Airliner_Ariplane .

log:Class_Deletion_Instance_1282056223671
      a       cho:Class_Deletion ;
      cho:hasChangedTarget vehicle:Truck ;
      cho:hasTimeStamp "1282056223671" ;
      cho:isPartOf
      log:Change_Set_Instance_2010_08_06_00_05_04;
      cho:isSubClassOf vehicle:Landcraft.
```

```
log:Property_Addition_Instance_1282057555140
      a       cho:Property_Addition ;
      cho:hasChangedTarget vehicle:objectProperty_2 ;
      cho:hasPropertyType owl:ObjectProperty ;
      cho:hasTimeStamp "1282057555140" ;
      cho:isPartOf
      log:Change_Set_Instance_2010_08_06_00_05_04.


log:Property_Renaming_1282057565796
      a       cho:Property_Renaming ;
      cho:hasChangedName "landsOn";
      cho:hasOldName "objectProperty_2";
      cho:hasPropertyType owl:ObjectProperty ;
      cho:hasTimeStamp "1282057565796";
      cho:isPartOf
      log:Change_Set_Instance_2010_08_06_00_05_04.


log:Domain_Addition_Instance_1282057572968
      a       cho:Domain_Addition ;
      cho:hasChangedTarget vehicle:landsOn;
      cho:hasDomain vehicle:Helicopter ;
      cho:hasPropertyType owl:ObjectProperty ;
      cho:hasTimeStamp "1282057572968";
      cho:isPartOf
      log:Change_Set_Instance_2010_08_06_00_05_04.

log:Range_Addition_Instance_1282057580015
      a       cho:Range_Addition ;
      cho:hasChangedTarget vehicle:landsOn ;
      cho:hasPropertyType owl:ObjectProperty ;
      cho:hasRange "Sea_Vessel";
      cho:hasTimeStamp "1282057580015";
      cho:isPartOf
      log:Change_Set_Instance_2010_08_06_00_05_04.

log:Property_Deletion_Instance_1282057717953
      a       cho:Property_Deletion ;
      cho:hasChangedTarget vehicle:carColor ;
      cho:hasPropertyType owl:ObjectProperty ;
      cho:hasTimeStamp "1282057717953";
      cho:isPartOf
      log:Change_Set_Instance_2010_08_06_00_05_04.
.
.
.
.
```

Figure 8, Representation of *Change_Set* instance from Change History Log (CHL) with corresponding change entries, reason for O_2 (given in Figure 3) evolution to O_2' using Change History Ontology [22].The changes are represented using N3 notation. Here log is prefix for Change History Log, cho is prefix for Change History Ontology, and vehicle is used as prefix for the Vehicle ontology.

ontology. The notion of *Change_Set* has been introduced that bundles all the ontology changes together that result in its evolution from one state to another. So this separate *Change_Set* instance helps in proper reformulation of query for required version/state of ontology (Figure 8 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 9).

Consider a SPARQL query (given below) written over Ontology O_2, shown in Figure 9, using a web service. This query will retrieve all the instances of class *Car* form Ontology O_2 in descending order of their names.

```
SELECT ?car ?carName
WHERE { ?car a :Car .
?car :hasName ?carName }
ORDER BY DESC(?carName)
```

Let's consider that Ontology O_2 evolved to another state Ontology O_2'. Now the same query from the same web service will not be able to extract the required information from Ontology O_2'. For this purpose, we need to reformulate the query for the newer state of ontology. The query is reformulated using change history information extracted from CHL using SPARQL queries given below.

```
SELECT ?changeSet ?timeStamp
WHERE { ?changeSet a :Change_Set .
?changeSet :hasTimestamp ?timeStamp }
ORDER BY DESC(?timeStamp)
```

The above query extracts the *Change_Set* instance where the changes are stored. The query below will extract changes using the *Change_Set* instance information from CHL.

```
SELECT ?change ?changedName ?oldName
WHERE { ?change :isPartOf "changeSet" .
?change :hasOldName ?oldName .
?change :hasChangedName ?changedName .
FILTER regex(?oldName, "Car") }
```

After extracting the information from CHL using above queries, the first query is rewritten as given below. This changed query will now get the required information from the evolved state of ontology O_2'.

```
SELECT ?car ?carName
WHERE { ?car a :Light_Vehicle .
?car :hasName ?carName }
ORDER BY DESC(?carName)
```

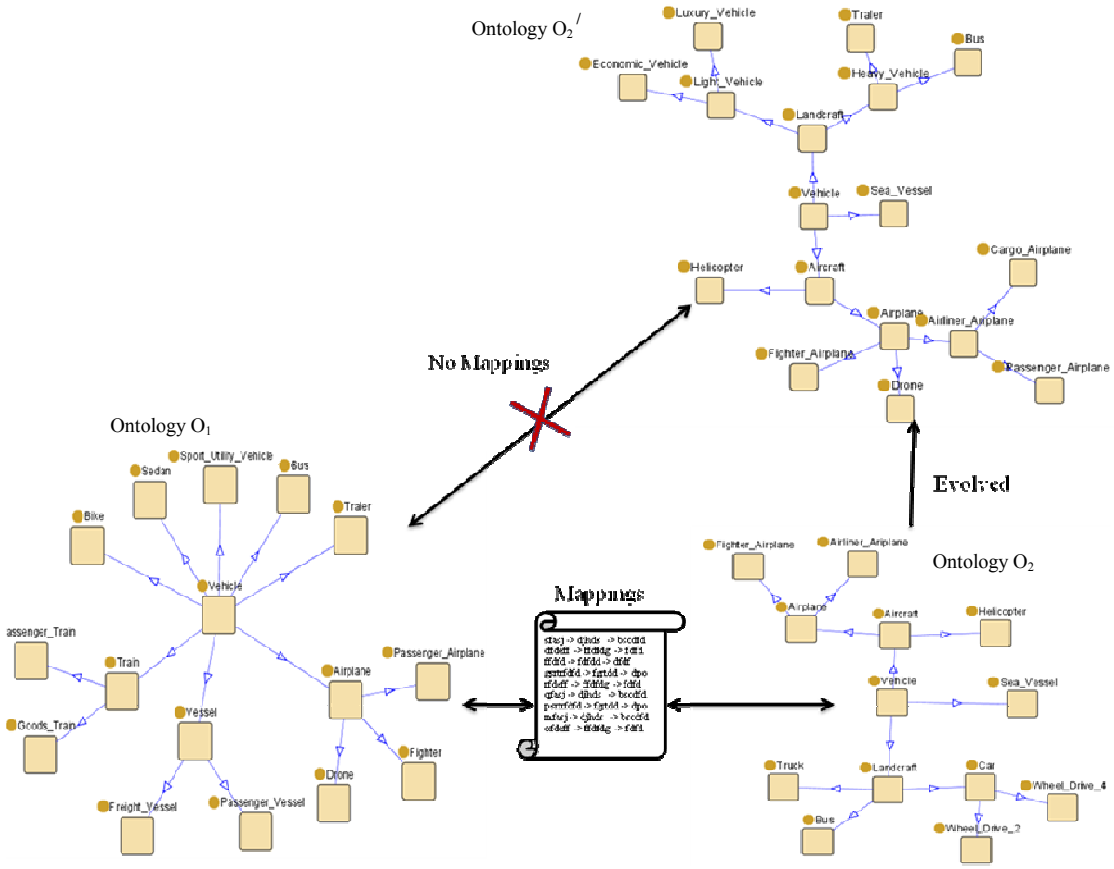

Figure 9, Ontology O_1 and O_2 having mappings, Ontology O_1 have evolved from state O_1 to state O_1', so the previous mappings are no more reliable as there are different changes introduced in O_1'.

5.7 Rebuilding Ontology Mappings

Mappings among systems, web services, and ontologies are required to translate query and/or share information [10, 16, 34, 35, and 54]. When ontology evolves from one consistent state to another then its mappings with the other ontologies are no more reliable and query execution and information exchange over such mappings among web services will produce unpredictable results. So when an ontology, having mappings established with other ontology, evolves then there is also a need for re-establishment of these stalled mappings.

There is no such solution for reengineering the broken mappings among the evolved ontologies except to completely re-establish the mappings. Re-establishing the mappings among small ontologies is not a problem, 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

[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

process. To solve this problem in a time efficient manner, we proposed that *Change History Log (CHL)* [22] containing all the changes (reason for ontology evolution) can play an important role here.

Consider two ontologies are mapped and they exchange information based on the established mappings. Now one or both the ontologies change (evolve) to another state. In this case the already existing mappings are not reliable and also become stale. The mappings between these two ontologies thus need to evolve with the evolving ontologies in order to be up to date. The scenario is discussed in two cases: (1) When One of the mapped ontologies evolves, (2) When both ontologies evolve from one consistent state to another. In both cases, the mappings also need to evolve to accommodate for the changed resources to eliminate the staleness from the already established mappings and facilitate information exchange among web services.

To reconcile the mappings in a time efficient manner and remove the stalled mappings, we proposed using the CHL entries for both the ontologies. It helps to identify the changed resources from both ontologies, establish mappings for these changed resources and update the old mappings. We only need to extend the method for calculating Semantic Affinity (SA) by incorporating the change information from CHL. Signature for SA is given below.

$SA(C_1, \Delta_1, C_2, \Delta_2, \psi)$

- C_1 Resource from ontology O_1
- Δ_1 Change information from CHL of Ontology O_1
- C_2 Resource from Ontology O_2
- Δ_2 Change information from CHL of Ontology O_2
- ψ User defined threshold for resource match

Though this proposed process for reconciliation of mapping reduces time, but it raises the concern for the accuracy of the reestablished mappings. There is a need to come up with a technique that should not only reduce the time for mapping reconciliation but should also produce the same amount of accurate mappings that systems like H-Match, MARRA, and Falcon [10, 16, and 35] generate.

5.8 Collaborative Ontology Engineering

The ontology as by definition is shared, so changes made to any instance of ontology should also be reflected to all other instance of the ontology. To support the concept of collaborative ontology engineering, a sophisticated and formal structure for change management is required that can bundle the changes of a specific change session and later propagate the changes to all the other instances as shown in Figure 10 where the changes of O_w and O_x are propagated to O_1, O_2, and O_3. For collaborative ontology engineering where ontology engineers are on remote locations and engineering an ontology then the concept of ontology change management becomes very important. It should also facilitate the change

in ontology based on the time order and avoid any time relevant conflicts. The changes are also propagated to each instance of ontology to keep the instances synchronized. The task of collaborative ontology engineering and change management is an important challenge of modern days that needs proper attention [42].

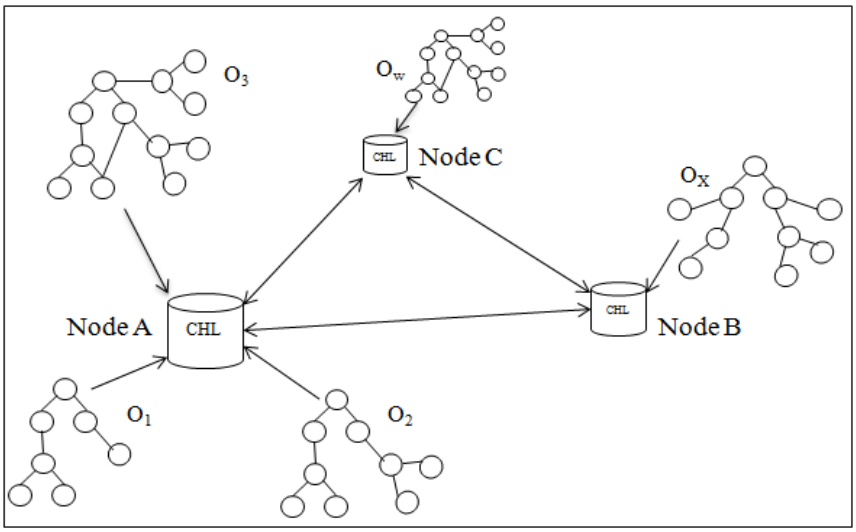

Figure 10: This figure represents the scenario of collaborative ontology engineering and importance of change management.

6.0 Conclusion

Ontology evolution is a collaborative process and incorporates work from related fields like ontology matching, merging, integration, versioning, and reasoning. We discussed different changes that result in ontology evolution, tools that support evolution, and the approaches followed for ontology evolution by the research community with their comparative analysis. To automate the process of ontology evolution, some of the unsolved problems were highlighted, and possible solutions for the highlighted problems were also suggested. The effects of ontology evolution on dependent data, application, information systems, ontology, and services based on ontology were discussed. We also proposed possible solutions for these after effects of ontology evolution and also referred to our developed solutions for some of these problems. We believe that the challenges identified in this article are of critical nature and addressing them would make the operation of Web Services, Applications, and Systems that use ontology smoother. Currently, we are working on improving the performance of our reestablishment of ontology mapping technique. The preliminary results are very encouraging in terms of performance but the accuracy is still the main focus in addition to maintaining the same level of performance. Implementation of change prediction in evolving ontologies is also in pipeline.

Acknowledgment

Acknowledgment will be added later.

References

1. J. Akahani, K. Hiramatsu, and T . Satoh. “Approximate query reformulation based on hierarchical ontology mapping. In proc. Of International Workshop on Semantic Web Foundations and Application Technologies (SWFAT), pages43–46, 2003.
2. M. Baldauf, S. Dustdar, and F. Rosenberg. “A survey on context-aware systems”. Int. Journal of Ad Hoc and Ubiquitous Computing, 2(4):263–277, June 2007.
3. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens, “OilEd: A Reasonable Ontology Editor for the Semantic Web” Proceedings of the 24th German / 9th Austrian Conference on Artificial Intelligence (KI-01), 2001.
4. T.Berners-Lee,J.Hendler,O.Lassila.TheSemanticWeb.ScientificAmerican,284(5):34-43,2001.
5. E. Brynjolfsson, H. Mendelson. 1993. Information systems and the organization of modern enterprise. *J.* organ. Comput. 3(3) 245–255, 1993.
6. P. B. Bhat, C. S. Raghavendra, and V. K. Prasanna. “Efficient Collective Communication in Distributed Heterogeneous Systems”. In Proc. 19th IEEE International Conference on Distributed Computing Systems (ICDCS), pages15–24, Austin, TX, June1999.
7. R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”. Future Generation Computing Systems, June 2009.
8. 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.
9. H. Chen, T. Finin, and A. Joshi, “An ontology for context-aware pervasive computing environments”, *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, Vol. 18, No. 3, pp.197–207, 2004.
10. 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.
11. CIDOC Conceptual Reference Model, http://cidoc.ics.forth.gr/
12. 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.
13. G. Flouris,and D. Plexousakis, “Handling Ontology Change: Survey and Proposal for a Future Research Direction” Technical Report FORTH-ICS/TR-362, 2005.
14. T. Gabel, Y. Sure, and J. Voelker. “KAON – ontology management infrastructure”. D3.1.1.a, SEKT Project: Semantically Enabled Knowledge Technologies, March 2004.
15. J. Heflin, J. Hendler, and S. Luke. Coping with changing ontologies in a distributed environment. In In Proceedings of the Workshop on Ontology Management of the 16th National Conference on Artificial Intelligence (AAAI), Florida, USA, July 18-22 1999. AAAI Press.
16. W. Hu and Y. Qu. “Falcon-AO: A Practical Ontology Matching System”, Journal of Web Semantics, 2007.
17. S. M. Huff, R. A. Rocha, B. E. Bray, H. R. Warner, P.J. Haug. “Anevent model of medical information representation. “ JAMIA.1995; 2: 116-134, 1995.
18. P. Haase, Y. Sure. “State of the Art on Ontology Evolution”, D3.1.1.b, SEKT Project: Semantically Enabled Knowledge Technologies, August 2004.
19. P. Haase, L. Stojanovic, “Consistent Evolution of OWL Ontologies”. In Proceedings of the 2nd European Semantic Web Conference (ESWC), 2005.
20. T. A. Halpin, “Information Modelling and Relational Databases: From Conceptual Analysis to Logical Design”. Morgan Kaufman Publishers, San Francisco, 2001.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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.
26. A. M. Khattak, J. Mustafa, N. Ahmed, K. Latif, S. Khan, "Intelligent Search in Digital Documents," Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on, vol. 1, pp. 558-561, 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2008.
27. A. M. Khattak, Z. Pervez, K. Latif, S. Y. Lee, Time efficient reconciliation of mappings in dynamic web ontologies, Knowledge-Based Systems, Available online 21 April 2012, ISSN 0950-7051, 10.1016/j.knosys.2012.04.016.
28. 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.
29. A. M. Khattak, Z. Pervez, S. Y. Lee, Y. K. Lee, “After Effects of Ontology Evolution”, The 5th International Conference on Future Information Technology (FutureTech10), Busan Korea, May, 2010.

30. A. M. Khattak, Z. Pervez, J. Sarkar, Y. K. Lee, “Service Level Semantic Interoperability”, International Workshop on Computing Technologies and Business Strategies for u-Healthcare (CBuH 2010)”, Seoul, Korea, July 2010.

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

32. H. Liu, C. Lutz, M. Milicic, F. Wolter, “Updating Description Logic ABoxes” Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06), 2006.

33. M. Lenzerini, “Data integration: a theoretical perspective”. In Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (Madison, Wisconsin, June 03 - 05, 2002). PODS '02. ACM, New York, NY, 233-246. DOI= http://doi.acm.org/10.1145/543613.543644.

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

35. A. Maedche, B. Motik, N. Silva, and R. Volz, “MAFRA - A MApping FRAmework for Distributed Ontologies.” In Proceedings of the 13th international Conference on Knowledge Engineering and Knowledge Management, pp 235-250, London, 2002.

36. M. Martin, S. Paolucci, M. McIlraith, D. Burstein, D. McDermott, B. McGuinness, T.Parsia, M. Payne, M. Sabou, N. Solanki, Srini-vasan, and K. Sycara,“Bringing semantics to web services :The OWL-S approach,” presented at the First Int.Workshop on Semantic Web Services and Web Process Composition (SWSWPC2004), SanDiego ,CA, 2004.

37. G. A. Miller. “WordNet: An on-line lexical database”. International Journal of Lexicography, 3(4):235–312. Special Issue. 1990.

38. 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.

39. N. Noy, R. Fergerson, and M. Musen, “TheKnowledgeModelofProtege-2000: Combining Interoperability and Flexibility” Proceedings of the 12th International Conferenceon Knowledge Engineering and Knowledge Management: Methods, Models, and Tools (EKAW-00), pp.17-32. 2000.

40. 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.

41. 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.

42. R. Palmaa, O. Corchoa, A. Gomez-Pereza, and P. Haase. A holistic approach to collaborative ontology development based on change management. Journal of Web Semantics, 9(3), 2011.

43. M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara, “Semantic matching of web services capabilities,” in proceedings of International Semantic Web Conference (ISWC-2002), 2002.

44. 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.

45. C. Preist, “A conceptual architecture for semantic web services.” *3rd International Semantic Web Conference* (*ISWC2004*). Springer Verlag. 2004.

46. D. D. Roure, N. R. Jennings, and N. R. Shadbolt, “The Semantic Grid: Past, present and future.” In *Proceedings of the IEEE 93*, 3 (2005), 669–681.

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

48. Y. Sure, J. Angele, and S. Staab, “OntoEdit: Multi faceted Inferencing for Ontology Engineering” Journal on Data Semantics,1(1),pp.128-152. 2003.

49. 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.

50. G. Stumme and A. Mädche. “FCA-merge: bottom-up merging of ontologies. In *Proc. 17th IJCAI, Seattle (WA US)*, pages 225–230, 2001.

51. O. Udrea, L. Getoor, and R. J. Miller, “Leveraging data and structure in ontology integration”. In Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data (Beijing, China, June 11 - 14, 2007). SIGMOD '07. ACM, New York, NY, 449-460. DOI= http://doi.acm.org/10.1145/1247480.1247531.

52. M. VanAntwerp and G. Madey. Warehousing, mining, and querying open source versioning metadata. Journal on Metadata Semantics, 2008.

53. M. Volkel and T. Groza. “SemVersion: RDF-based ontology versioning system”. In Proceedings of the IADIS International Conference WWW/Internet 2006 (ICWI 2006), 2006.

54. P. Wang and B. Xu. Lily: Ontology alignment results for oaei 2009. In Ontology Matching of 8th International Semantic Web Conference (ISWC), Washington DC, USA, October 25-29 2009. LNCS.

55. R. Wasserman. The problem of change. Philosophy Compass, 1(1), 2006.

56. 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.