Ontology Evolution and Challenges[*]

ASAD MASOOD KHATTAK[1], RABIA BATOOL[1], ZEESHAN PERVEZ[1],
ADIL MEHMOOD KHAN[2] AND SUNGYOUNG LEE[1,+]
[1]*Department of Computer Engineering*
Kyung Hee University
Yongin, 446-701 Korea
[2]*Division of Information and Computer Engineering*
Ajou University
Gyeonggi-do, 443-749 Korea

Information semantics and semantic interoperability among applications, systems, and 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 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 introduces some unfolded challenges in the field of ontology evolution, which must be tackled to complete the process automatically. Moreover, the new changes 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, knowledge management applications, ontology, ontology change, ontology change management, ontology evolution

1. INTRODUCTION

The fundamental aspect of information exchange among applications, systems, and 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 [1], exchange of information among clinics [2], and among heterogeneous systems [3]. To make this possible, we need to carefully model the domain knowledge while preserving its semantics [4, 5].

Ontology provides formal structure with semantics about how an expert perceives

Received July 29, 2011; revised May 24, 2012; accepted December 6, 2012.
Communicated by Elena García-Barriocanal.
[+] Corresponding author.
[*] This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2012-(H0301-12-2001)). This work was supported by the Industrial Strategic Technology Development Program (10035348, Development of a Cognitive Planning and Learning Model for Mobile Platforms) funded by the Ministry of Knowledge Economy (MKE, Korea). This research was supported by the new faculty research fund of Ajou University. This paper is supported by sabbatical year of Kyung Hee University in 2010.

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 [6]. Ontologies are complex in nature and often large structured. Their development and maintenance incorporates research areas like: evolution, versioning, merging, and integration where these are fundamentally different [7].

Different convergence technologies like: Semantic Web Services [8, 9], Context-aware Search Engines [10], Software Agents [11], and Semantic Grid [12] use ontology for their customized needs [13]. Systems using context-aware information (modeled using ontology) offer opportunities for applications, services, application developers, and end users by gathering context information. The modeled information facilitates in adapting systems behavior according to application and end user customized needs. Especially in combination with mobile devices, this modeled information is of high importance that increases usability of information and applications on top of the modeled information tremendously [13]. Currently in use and most appreciate approach for information modeling, mediation, and integration is use of ontology in every aspect of data, application, system, service, and technology level integration and interoperability [4, 13-15].

Thus the use of ontology is increasing in Information Systems and Knowledge Sharing Systems, which in response increases the significance of ontology maintenance [5, 7, 16]. Ontology change results in evolution of ontology where ontology evolution is the change in expert's perception about the domain in view [17]. 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. 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, systems, services, and applications [5, 7, 19].

Ontology evolves from one consistent state to another [20] 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* [7, 15, 16, 21, 22]. Research on ontology evolution is being carried out by different researcher's groups, and their approaches overlap with each other [7, 16-19, 21, 22]. 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. 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 [23]. 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, 24, 25].

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 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 its effects of dependent service. Section 3 briefly discusses different types of ontology changes and change management activities to cope with these changes. Section 4 presents existing ontology evolution approaches 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 on the dependent data, systems, and technologies. Finally, we conclude our discussion in Section 6.

2. ONTOLOGY CHANGE A CASE STUDY

Ontology is being used by different convergent technologies for their customized needs, such as Context-aware Search Engines [10], Software Agents [11], and Semantic Grid [12]. In this section, we mainly focus on Semantic Web Services [8, 9, 26] (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.

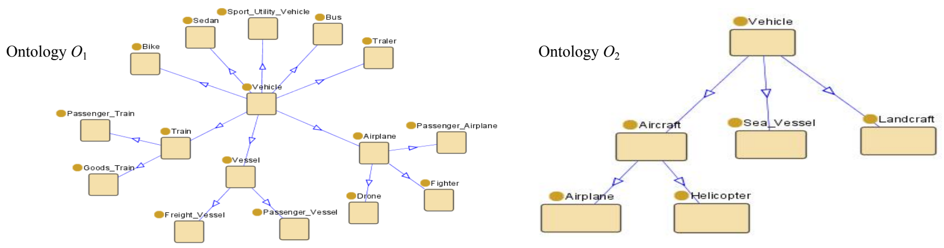


Fig. 1. Vehicle ontologies, Ontology O_1 and O_2 are used by different web services.

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 Fig. 1, is an extended version of the ontology used in [26]. Consider two web services that provide services about *Vehicle*. These services use the two ontologies given in Fig. 1 respectively, to fulfill 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 [26-29] 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 Fig. 2). To evolve the mappings and continue the sharing of information, proper maintenance of ontology changes and their reflection is necessary [7, 15, 30].

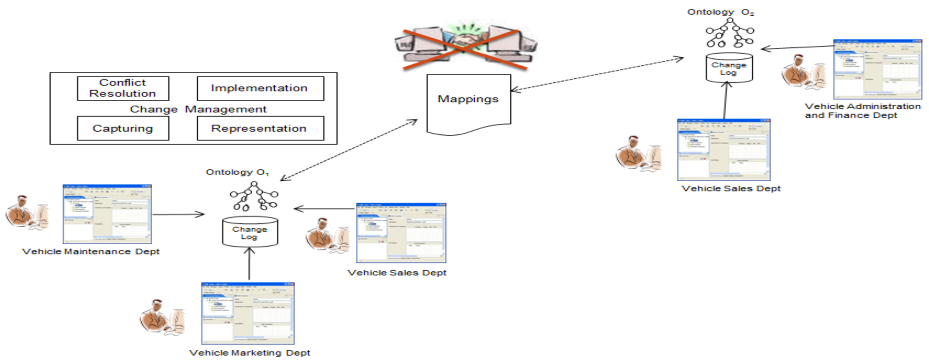


Fig. 2. Vehicle ontologies, Ontology O_1 and Ontology O_2 used for information sharing scenarios based on the established mappings. However, changes initiated from any department will make the mappings unreliable which will stop the sharing of information between the two ontologies. Change management module facilitates consistent evolution of ontology.

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 [7]. Consider the scenario given in Fig. 2 where the *Vehicle* ontology 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 change management module. 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. ONTOLOGY CHANGE MANAGEMENT ACTIVITIES

As explained in [7] 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. Different changes and their effects are mostly discussed in [19, 31]. Ontology change management deals with four main activities which are related and in some cases overlap but are fundamentally different activities.

- *Ontology evolution* is the process of modifying ontology in response to a certain change in the domain or its conceptualization [7, 16, 19, 22, 31].

- *Ontology versioning* is the ability to handle an evolving ontology by creating and managing its different versions [7, 16, 32, 33].
- *Ontology integration* is the process of composing an ontology on a particular subject from information found in two or more ontologies covering multiple domains [7, 16, 34].
- *Ontology merging* is the process of composing an ontology from information found in two or more ontologies covering highly overlapping or identical domains [7, 16, 35].

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 [19]. 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. Understanding change types is necessary to correctly handle explicit and implicit change requirements [18], and consequently understand the effects. 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 [19]. (1) Changes at class level (adding, deleting, updating, and renaming) correspond to changes that are directly related to 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 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. 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. Let's consider the Ontology O_2 given in Fig. 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. ONTOLOGY EVOLUTION APPROACHES

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 [7, 16, 22, 31]. 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 [36, 37]	• Mostly used for ontology creation • Often used for evolution and maintenance • Provides Merging, Integration, and Comparison • SparQL queries support	• Weak ontology change management • No facility for ontology recovery • Use third party services for consistency checking of ontology	Manual evolution support
KAON [38]	• Provides ontology editing services like Protégé • Provides environment for pre-evolution strategy making, avoid conflicts using deduce changes • 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 [39]	• Used for ontology engineering • Disallows inconsistency in ontology • Supports semi-automated ontology evolution	• No change logging facility • No facility for ontology recovery • Strict in its operations	Semi-auto matic support
OntoEdit [40]	• Used for ontology editing • More options than KAON for strategy making • Allows collaborative editing environment	• 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 Fig. 4, ontology). *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. This process is mainly related to the area of ontology debugging [7]. *Change Implementation and Verification:* where the complete change request is executed on the ontology and validation is performed to see if the requested changes made to ontology or not. All the applied changes are also logged into a repository and these logged changes are 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* [24]. *Ontology Population* is when we get new instances for concept that is already present in the ontology. Here only the new instance(s) of the concept 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 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.

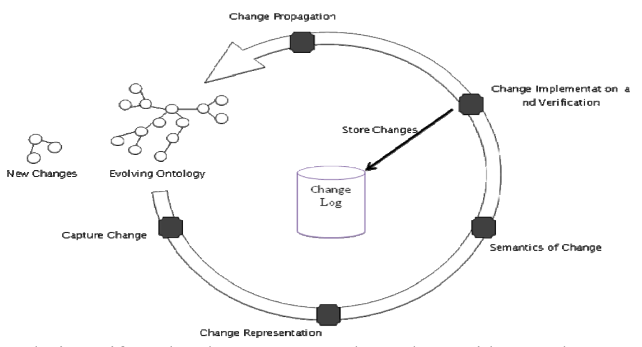


Fig. 3. Ontology Evolution Lifecycle takes source ontology along with new changes and implements the new changes to source ontology.

Initially, L. Stojanovic and B. Motik in [22] talked about the support that different ontology editors have, their limitations, and complexities, and usability issues of these tools for ontology evolution management. As ontology needs refinements, so it must be updated by making appropriate changes in it. Therefore, methods to cope with the changes that result from 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 for ontology evolution and propagation of these changes are discussed in [22]. 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. In [7, 19, 36, 38, 41, 49], new changes are specified by ontology engineer, whereas [16, 42-44] detect new changes between two different versions of ontologies using PromptDiff (Protégé plug-in), OntoView [45], and H-Match [27] algorithms. The systems discussed in [21, 31, 49] detect new changes mostly using WordNet and H-Match [27]. 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) [19], Change Log [44], and Change History Ontology (CHO) [21]. 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 [38] and OntoEdit [40] use a predefined resolution strategy, and Evolva of NeOn Toolkit [49] uses a customized run time strategy for conflict resolution.

After the implementation of the new changes, some systems [19, 21, 38, 44] provide the facility for change logging [23]. There is also a need to select an appropriate level of granularity for change item. A very simple level can cause side effects [22] while complex level can make problems during ontology recovery and can pass up the facility for item level recovery [23]. After the validation of implemented changes, they are propagated to the dependent data, applications, and services using push-based and pull-based [22] 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 and they might not need the updates. So, the pull-based approach is suitable, which excludes change propagation phase from ontology evolution procedure.

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 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 [22] 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 [22, 42], users manually created the requests for changes, whereas the conflicts were manually resolved by experts.
- In [31], author focused on discovery of new change 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 [21, 49] systems were the best matching resource(s) selection for the newly emerging change.

Table 2. Summary of ontology evolution approaches. Last column represents maturity level of the approach in terms of automation.

Approaches	Change Request	Change Representation	Conflict Resolution	Change Implementation	Change Propagation	Working
L. Stojanovic, *et al.* [22].	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, applitions, and services. Out dated instances are replaced.	User intervention required for system working
M. Klein, N. Noy, *et al.* [19, 36, 41]	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.* [38] (KAON)	Specified by ontology engineer	Formal representation of changes	Predefined strategies for conflict resolution.	Provides interface for user interaction and also logs the changes.	Propagation of changes to dependent artifacts.	Most parts of the system are working
P. Plessers, *et al.* [42]	Different versions of ontologies are used in this approach. Changes among different versions are represented formally.		After change implementation, it checks for inconsistencies and implement 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.* [43, 44]	Changes detected among two versions by using Prompt-Diff and OntoView [45], 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.* [42]	Use top-down manual and bottom-up automatic approach for change detection	Suggestions for the use of formal representation *i.e.*, using the change log representation	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.* [46]	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.* [31]	Changes are recognized automatically by analyzing domain artifacts. H-Match [27] and WordNet[1] [47] are used for 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.* [48]	New changes such as (change in single concept, group of concepts and concepts in a hierarchical structure) are detected automatically using H-Match [27] and WordNet. Change representation is provided by Change History Ontology (CHO) [21].		For conflict resolution KAON API [38] is used with some suggested extensions.	Changes are implemented atomically and after every change implementation, these are logged in CHL [21]. 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 [49]	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 evolution

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

- Inconsistency resolution is also amongst the most critical problems that needs attention before, during, and after evolution. The consistency is checked for; consistent modeling of new resources in presence of existing resources, consistency with the other side matching ontology, and consistency with the business rules of organization.
- In [21] the author proposed a training process for different deduced changes. But training a system for induced and 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 [25, 49].

5. OPEN CHALLENGES

In this section, we explain in detail the challenges that need to be solved for the purpose of achieving automated ontology evolution [24]. Afterwards, we discuss in detail some of the after effects of ontology evolution on the dependent data, applications, systems, services, and 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 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 services and ontology.

5.1 Change Detection

The first phase of automatic ontology evolution is to detect new changes. To detect changes among the newly emerging concept(s), various resource correspondence, difference, and matching [27, 43] techniques are applied. This helps in finding out the most appropriate position for the emerging concept in concept hierarchy [7]. 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 [21, 27] where the new concept should be inserted. However, there exist two different problems [24]:

- *Relevance Detection:* It means to calculate the relevance between the new concept and the existing concept(s). The currently used algorithms for difference, correspondence, and matching are presented in [27, 28, 43]. 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 Fig. 4, these algorithms [27, 28, 43] 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, 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*.

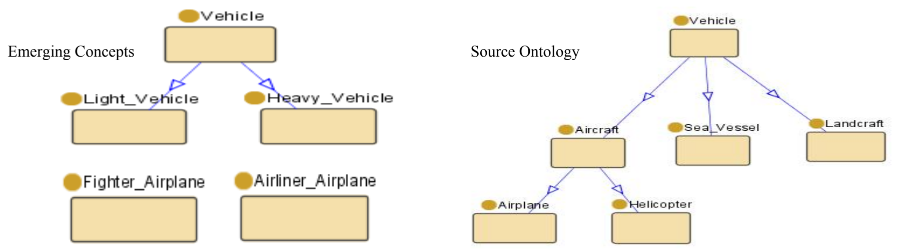


Fig. 4. Emerging concepts are the newly detected changes (discoveries) in domain and are the reasons for change in source ontology (*i.e.*, Vehicle Ontology in use of a web service).

- *Selection among Newly Detected Changes:* It is quite possible that more than one concept is related to the newly emerging concept, so which alternative should be selected? To understand this problem concentrate on Fig. 4, 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 [17]. These issues are still unsolved and need attention.

5.2 Conflict Analysis

Consistency of ontology after evolution is the most critical concern. 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 systems use expert intervention for resolving the conflicts [22, 31, 41, 42]. In KAON [38] and OntoEdit [40], 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 [38, 40]. 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. 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:* In [21], 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 Fig. 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).

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* [21]). Based on the above mentioned categories, we derive instances of class *Ontology-Change*, represented with the symbol Δ, using the following axioms, for details see [21].

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

Changes and the reasons for the 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 [21]. Changes of specific time interval are logged as one *Change_Set* (see Fig. 6), 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 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, and services, which could help in understanding the changes in ontology, data, application, and service behavior.

5.4 Ontology Recovery and Change Visualization

Proper maintenance of ontology 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 [38], Protégé [37], and OntoEdit [40] 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.

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.

We validated our proposed framework as an added component for the ontology editor (*i.e.*, Protégé) [23]. The recovery and visualization component, on top of all other components, should provide ontology recovery and visualization services. For details on these procedures and validation refer to [23]. 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.

5.5 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 [21] are atomic level (simple) changes that do provide lots of open space for change patterns. See Fig. 6, 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. Using these frequent patterns, we can also get a better understanding of development. Fig. 5 is an example of mining frequent patterns from some logged changes.

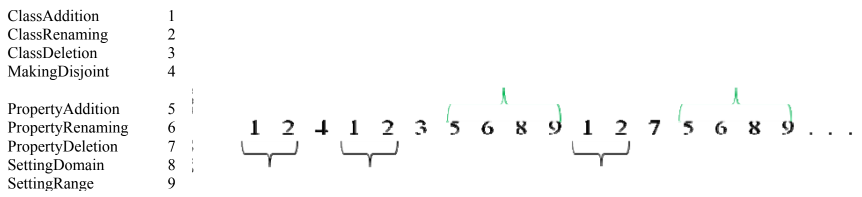


Fig. 5. Mining frequent change pattern from logged changes.

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

Fig. 6. Representation of *Change_Set* instance from CHL with corresponding change entries, reason for O_1 evolution to $O_1^{/}$ using CHO. The changes are represented using N3 notation.

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

5.6 Query Reformulation

Query written over one schema does not give correct results when executed over another schema [50], so it needs to be reformulated. Same is true for ontology, so when ontology evolves then the query written over previous state needs to be reformulated to extract the required results from the evolved ontology [51]. The author in [51] proposed a five phase query reformulation procedure for evolved ontologies. The main modules of the procedure are: *capture*, *instantiate*, *analyze*, *update*, and *respond* (for details please refer to [51]). They evaluated the system using two different versions of CRM[2] [52] ontology. The main idea behind this work is to maintain the changes in a repository and later used these changes for query reformulation.

This system 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 is hard to extract the changes from the log that corresponds to a particular state of ontology. In [21], 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 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 state of ontology (Fig. 6 is an example of *Change_Set* instance that cause the ontology O_1 evolved to $O_1^{/}$ state, as shown in Fig. 7).

Consider a SPARQL query (given below) written over Ontology O_1 shown in Fig 7. This query will retrieve all the instances of class *Car* form Ontology O_1 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_1 evolved to another state $O_1^{/}$. Now the same query will not be able to extract the required information from Ontology $O_1^{/}$. For this purpose, we need to reformulate the query for the newer state. 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 $O_1^{/}$.

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

5.7 Rebuilding Ontology Mappings

Mappings are required to translate query and/or share information [27-29, 51, 53]. When ontology evolves then its mappings with the other ontologies are no more reliable and query execution and information exchange over such mappings will produce unpredictable results. So there is also a need for re-establishment of these stalled mappings.

There is no such solution for re-engineering 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], then re-establishment of mappings among such ontologies is a time consuming process. To solve this problem in a time efficient manner, we believe *Change History Log* (*CHL*) [21] containing all the changes can play an important role.

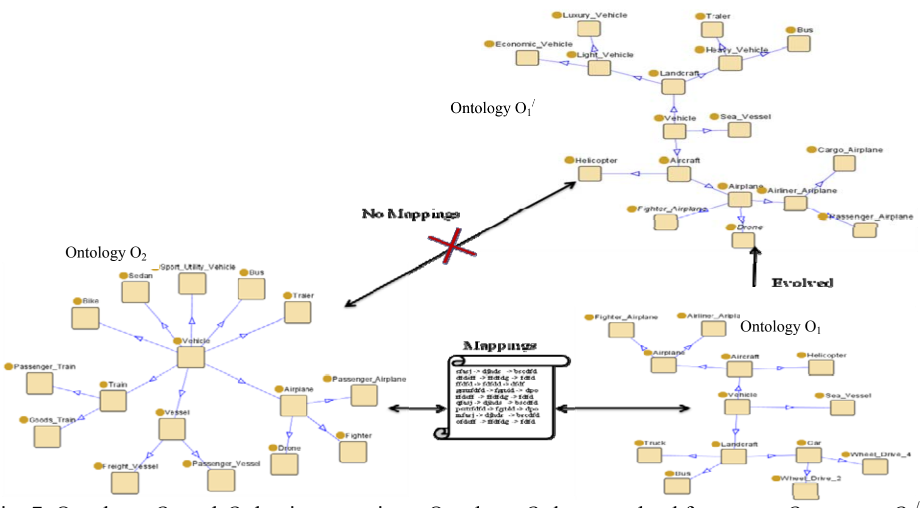


Fig. 7. Ontology O_1 and O_2 having mappings, Ontology O_1 have evolved from state O_1 to state $O_1^{/}$, so the previous mappings are no more reliable as there are different changes introduced in $O_1^{/}$.

Consider two ontologies exchange information based on the established mappings. Now one or both the ontologies change (evolve). 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

[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

need to evolve to accommodate for the changed resources to eliminate the staleness from the already established mappings and facilitate information exchange.

To reconcile the mappings in a time efficient manner and remove the stalled mappings, we proposed using the CHL entries. 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.

$$SA(C_1,\Delta_1,C_2,\Delta_2,\psi)\begin{cases} C_1 \text{ Resource from ontology } O_1 \\ \Delta_1 \text{ Change information from CHL of Ontology } O_1 \\ C_2 \text{ Resource from ontology } O_2 \\ \Delta_2 \text{ Change information from CHL of Ontology } O_2 \\ \psi \text{ User defined threshold for resource match} \end{cases}$$

Though this proposed process for reconciliation of mapping reduces time, but it raises the concern for the accuracy of the re-established 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 [27-29] 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 instances 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 Fig. 8 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 in-

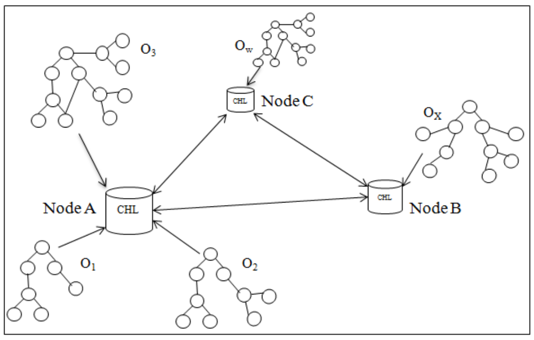


Fig. 8. Collaborative ontology engineering and importance of change management.

stances synchronized. The task of collaborative ontology engineering and change management is an important challenge of modern days that needs proper attention [54].

6. 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, applications, 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 re-establishment of ontology mapping technique. However, the accuracy is still the main focus in addition to maintaining good performance. Change prediction in evolving ontologies is also in pipeline.

REFERENCES

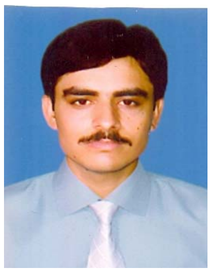
1. E. Brynjolfsson and H. Mendelson, "Information systems and the organization of modern enterprise," *Journal of Organizational Computing*, Vol. 3, 1993, pp. 245-255.
2. S. M. Huff, R. A. Rocha, B. E. Bray, H. R. Warner, and P. J. Haug, "An event model of medical information representation," *Journal of the American Medical Informatics Association*, Vol. 2, pp. 116-134.
3. P. B. Bhat, C. S. Raghavendra, and V. K. Prasanna, "Efficient collective communication in distributed heterogeneous systems," in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, 1999, pp. 15-24.
4. T. A. Halpin, *Information Modelling and Relational Databases: From Conceptual Analysis to Logical Design*, Morgan Kaufman Publishers, San Francisco, 2001.
5. R. Wasserman, "The problem of change," *Philosophy Compass*, Vol. 1, 2006, pp. 1-10.
6. T. B. Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, Vol. 284, 2001, pp. 34-43.
7. G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou, "Ontology change: Classification and survey," *Knowledge Engineering Review*, Vol. 23, 2008, pp. 117-152.
8. M. Martin, S. Paolucci, M. McIlraith, and *et al.*, "Bringing semantics to web services: The OWL-S approach," in *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition*, 2004, pp. 117-152.
9. C. Preist, "A conceptual architecture for semantic web services," in *Proceedings of*

the 3rd International Semantic Web Conference, 2004, pp. 395-409.

10. A. M. Khattak, J. Mustafa, N. Ahmed, K. Latif, and S. Khan, "Intelligent search in digital documents," *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 1, 2008, pp. 558-561.
11. 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, 2004, pp. 197-207.
12. D. D. Roure, N. R. Jennings, and N. R. Shadbolt, "The semantic grid: Past, present and future," in *Proceedings of the IEEE*, Vol. 93, 2005, pp. 669-681.
13. M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 2, 2007, pp. 263-277.
14. M. Lenzerini, "Data integration: A theoretical perspective," in *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2002, pp. 233-246.
15. J. Heflin, J. Hendler, and S. Luke, "Coping with changing ontologies in a distributed environment," in *Proceedings of the Workshop on Ontology Management of the 16th National Conference on Artificial Intelligence*, 1999, pp. 74-79.
16. G. Flouris and D. Plexousakis, "Handling ontology change: Survey and proposal for a future research direction," Technical Report FORTH-ICS/TR-362, 2005.
17. N. F. Noy and M. Klein, "Ontology evolution: Not the same as schema evolution," *Knowledge and Information System*, Vol. 6, 2004, pp. 428-440.
18. P. Haase and Y. Sure, "State of the art on ontology evolution," D3.1.1.b, SEKT Project: Semantically Enabled Knowledge Technologies, August 2004.
19. M. Klein, "Change management for distributed ontologies," Ph.D. Thesis, Department of Computer Science, Vrije University, Amsterdam, 2004
20. P. Haase and L. Stojanovic, "Consistent evolution of OWL ontologies," in *Proceedings of the 2nd European Semantic Web Conference*, 2005, pp. 182-197.
21. A. M. Khattak, K. Latif, S. Khan, and N. Ahmed, "Managing change history in web ontologies," in *Proceedings of the 4th International Conference on Semantics*, *Knowledge and Grid*, 2008, pp. 347-350.
22. L. Stojanovic, A. Madche, B. Motik, and N. Stojanovic, "User driven ontology evolution management," in *Proceedings of European Conference on Knowledge Engineering and Management*, 2002, pp. 285-300.
23. A. M. Khattak, K. Latif, S. Y. Lee, Y. K. Lee, M. Han, and H. Il-Kim, "Change tracer: Tracking changes in web ontologies," in *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence*, 2009, pp. 449-456.
24. A. M. Khattak, K. Latif, S. Y. Lee, and Y. K. Lee, "Ontology evolution: A survey and future challenges," in *Proceedings of the 2nd International Conference on u- and e-Service*, *Science and Technology*, 2009, pp. 68-75.
25. A. M. Khattak, Z. Pervez, S. Y. Lee, and Y. K. Lee, "After effects of ontology evolution," in *Proceedings of the 5th International Conference on Future Information Technology*, 2010, pp. 1-6.
26. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *Proceedings of International Semantic Web Conference*, 2002, pp. 333-347.

27. S. Castano, A. Ferrara, and S. Montanelli. "Matching ontologies in open networked systems," *Techniques and Applications*, *Journal on Data Semantics*, Vol. V, 2006, pp. 25-63.
28. W. Hu and Y. Qu. "Falcon-AO: A practical ontology matching system," *Journal of Web Semantics*, Vol. 6, 2008, pp. 237-239.
29. 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*, 2002, pp. 235-250.
30. A. M. Khattak, Z. Pervez, K. Latif, and S. Y. Lee, "Time efficient reconciliation of mappings in dynamic web ontologies," *Knowledge-Based Systems*, Vol. 35, 2012, pp. 369-374.
31. S. Castano, A. Ferrara, and G. Hess, "Discovery-driven ontology evolution," *The Semantic Web Applications and Perspectives*, *3rd Italian Semantic Web Workshop*, 2006.
32. M. V. Antwerp and G. Madey, "Warehousing, mining, and querying open source versioning metadata," *Journal on Metadata Semantics*, 2008.
33. M. Volkel and T. Groza. "SemVersion: RDF-based ontology versioning system," in *Proceedings of the IADIS International Conference WWW/Internet*, 2006, pp. 195-202.
34. O. Udrea, L. Getoor, and R. J. Miller, "Leveraging data and structure in ontology integration," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2007, pp. 449-460.
35. G. Stumme and A. Mädche, "FCA-merge: bottom-up merging of ontologies," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, Vol. 1, 2001, pp. 225-230.
36. N. F. Noy, A. Chugh, W. Liu, and M. A. Musen, "A framework for ontology evolution in collaborative environments," in *Proceedings of International Semantic Web Conference*, 2006, pp. 544-558.
37. N. Noy, R. Fergerson, and M. Musen, "The knowledge model of Protégé – 2000: Combining interoperability and flexibility" in *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management: Methods*, *Models*, *and Tools*, 2000, pp. 17-32.
38. T. Gabel, Y. Sure, and J. Voelker, "KAON – ontology management infrastructure," D3.1.1.a, SEKT Project: Semantically Enabled Knowledge Technologies, March 2004.
39. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens, "OilEd: A reasonable ontology editor for the semantic web," in *Proceedings of the 24th German/9th Austrian Conference on Artificial Intelligence*, 2001, pp. 396-408.
40. Y. Sure, J. Angele, and S. Staab, "OntoEdit: Multifaceted inferencing for ontology engineering" *Journal on Data Semantics*, Vol. 1, 2003, pp. 128-152.
41. M. Klein and N. F. Noy, "A component-based framework for ontology evolution," in *Proceedings of the Workshop on Ontologies and Distributed Systems*, CEUR-WS, Vol. 71, 2003.
42. P. Plessers and O. de Troyer, "Ontology change detection using a versioning log," in *Proceedings of the 4th International Semantic Web Conference*, 2005, pp. 578-592.
43. D. Oberle, R. Volz, B. Motik, and S. Staab, "An extensible ontology software envi-

ronment," in *Handbook on Ontologies* (*Series of International Handbooks on Information Systems*), Springer, 2004, pp. 311-333.
44. D. Rogozan and G. Paquette, "Managing ontology changes on the semantic web," in *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*, 2005, pp. 430-433.
45. M. Klein, A. Kiryakov, D. Ognyanov, and D. Fensel, "Finding and characterizing changes in ontologies," in *Proceedings of the 21st International Conference on Conceptual Modeling*, 2002, pp. 79-89.
46. H. Liu, C. Lutz, M. Milicic, and F. Wolter, "Updating description logic ABoxes" in *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning*, 2006, pp. 46-56.
47. G. A. Miller, "WordNet: An on-line lexical database," *International Journal of Lexicography*, Vol. 3, 1990, pp. 235-312.
48. A. M. Khattak, K. Latif, S. Y. Lee, Y. K. Lee, and T. Rasheed, "Building an integrated framework for ontology evolution management," in *Proceedings of the 12th Conference on Creating Global Economies through Innovation and Knowledge Management*, 2009, pp. 55-60.
49. F. Zablith, "Ontology evolution: A practical approach," poster, in *Proceedings of Workshop on Matching and Meaning at Artificial Intelligence and Simulation of Behavior*, 2009.
50. J. Akahani, K. Hiramatsu, and T. Satoh, "Approximate query reformulation based on hierarchical ontology mapping," in *Proceedings of International Workshop on Semantic Web Foundations and Application Technologies*, 2003, pp. 43-46.
51. Y. D. Liang, "Enabling active ontology change management within semantic web-based applications" Mini Ph.D. Thesis, Electronics and Computer Science, University of Southampton, 2006.
52. CIDOC Conceptual Reference Model, http://cidoc.ics.forth.gr/.
53. P. Wang and B. Xu, "Lily: Ontology alignment results for oaei," in *Proceedings of Ontology Matching of the 8th International Semantic Web Conference*, 2009.
54. 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*, Vol. 9, 2011, pp. 299-314.



Asad Masood Khattak received his Ph.D. in Computer Engineering from the Department of Computer Engineering at Kyung Hee University, Korea in August 2012. He is currently working as Assistant Professor at Department of Computer Engineering, Kyung Hee University, Korea. His research interests include data management, knowledge representation, change management, semantic web, and ontology.

Rabia Batool received her BS degree in Information Technology from National University of Sciences and Technology Pakistan in 2011. She is pursuing her MS in Biomedical Engineering from the Department of Biomedical Engineering at Kyung Hee University, Korea. Her research interests include natural language processing, tweet analysis, health informatics, and interoperability.

Zeeshan Pervez received his MS-IT degree from SEECS NUST, Pakistan in 2008 and Ph.D. in Computer Science from Kyung Hee University, Korea in 2012. His graduate research was focused on privacy-aware cloud based data sharing systems. His research interests include secure multiparty computation, encrypted data search, and fine-grained access control in untrusted domain.

Adil Mehmood Khan received his Ph.D. degree from the Department of Computer Engineering of Kyung Hee University, Korea in 2011. He is now working as Assistant Professor with the Division of Information and Computer Engineering, Ajou University, Korea. His research interest includes pattern recognition, signal processing, data mining, ubiquitous computing, and machine learning.

Sungyoung Lee received his MS and Ph.D. in Computer Science from Illinois Institute of Technology (IIT), Chicago, Illinois, USA in 1987 and 1991, respectively. He has been a Professor in the Department of Computer Engineering, Kyung Hee University, Korea since 1993. He is a Founding Director of the Ubiquitous Computing Laboratory, and has been the Director of the Neo-Medical Ubiquitous-Life Care Information Technology Research Center, Kyung Hee University since 2006. He is a member of ACM and IEEE.