Thesis for the Degree of Doctor of Philosophy

Managing Change History in Dynamic Web Ontologies

Asad Masood Khattak

Department of Computer Engineering

Graduate School

Kyung Hee University

Seoul, Korea

August, 2012

Managing Change History in Dynamic Web Ontologies

Asad Masood Khattak

Department of Computer Engineering

Graduate School

Kyung Hee University

Seoul, Korea

August, 2012

Managing Change History in Dynamic Web Ontologies

by

Asad Masood Khattak

Advised by

Professor Sungyoung Lee

Submitted to the Department of Computer Engineering
and the Faculty of the Graduate School of
Kyung Hee University in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Dissertation Committee:

Professor Tae-Choong Chung, Ph.D. . . . . .

Professor Brian J. d'Auriol, Ph.D. . . . . .

Professor Kyung Mo Park, Ph.D. . . . . .

Professor Guan Donghai, Ph.D. . . . . .

Professor Sungyoung Lee, Ph.D. . . . . .

Managing Change History in Dynamic Web Ontologies

by

Asad Masood Khattak

Submitted to the Department of Computer Engineering
on July, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Knowledge constantly grows in scientific discourse and is revised over time by different stakeholders, either collaboratively or through institutionalized efforts. The body of knowledge gets structured and refined as the Communities of Practice concerned with a field of knowledge develop a deeper understanding of the issues. As a result, the knowledge model moves from a loosely clustered terminology to a semi-formal or even formal ontology. Change history management in such evolving knowledge models is a an important and challenging task. Different techniques (including change classification, representation, capturing, and logging) have been introduced in the research literature to solve the issue; however, these are limited in their scope. Ontology change management solutions must provide a consistent and coherent support to maintain and manage all the ontology changes. Moreover, the role of change history information becomes critical when an ontology engineer wants to review the ontology evolution history. A storage structure for such information is also crucial for effective retrieval. A single change also makes the existing mappings between ontologies unreliable. In addition, the re-establishment of mapping is also a time consuming process. To handle these challenges, a comprehensive solution is required that must address various multi-faceted issues, such as ontology change history management and traceability, recovery, visualization of change effects, keeping the evolving ontology in a consistent state, mapping reconciliation between evolving ontologies, query reformulation, and collaborative ontology engineering.

This research introduces a change history management framework for evolving ontologies. It is a comprehensive and methodological framework for managing issues related to change management in evolving ontologies, such as versioning, provenance, consistency, recovery, change representation, mapping reconciliation, and visualization. The Change history log is central to the proposed framework and is supported by a seman-

tically rich and formally sound change representation scheme known as Change History Ontology (CHO). Changes are captured and then stored in the Change History Log (CHL) in conformance with the CHO. Changes in resources consequently make the existing mappings between ontologies unreliable and staled. This highlights the need for mapping evolution to eliminate discrepancies from the existing mappings. To re-establish the mappings between dynamic ontologies, CHL entries of changing ontologies are used to eliminate the saltness from the existing mappings. The framework for change capturing is implemented to work as a plug-in for ontology editor, such as Protege. For the mapping reconciliation procedure, the proposed scheme is plugged in with the existing mapping systems. The existing systems restart the complete mapping process which is time and memory consuming process. The proposed mapping reconciliation approach between the updated ontologies takes less time and use lesser memory as compared to the existing systems by only considering the changed resources to eliminate the staleness from the mappings.

The framework is implemented to work as a plug-in for ontology editors, such as Protege. The change capturing accuracy of the proposed system *Change Tracer* is compared with that of *Changes Tab*, *Version Log Generator* in Protege; *Change Detection*; and *Change Capturing* of NeOn Toolkit. The proposed system has shown better accuracy against the existing systems. A comprehensive evaluation of change logging is also validate using recovery operations with different versions of *SWETO Ontology*, *CIDOC CRM Ontology*, *OMV Ontology*, and *SWRC Ontology*. The performance, space usage, and accuracy of proposed mapping reconciliation procedure is compared with existing mapping systems, such as *FOAM*, *Falcon*, *H-Match*, *Prompt*, *Lily*, *AgreementMaker*, and *TaxoMap*. Overall time efficiency of 43% to 85%, reduced the runtime memory consumption in range of 41.11% to 58.43%, and accuracy range of 93.50% to 100% in comparison to existing mapping systems is achieved. The experimental results and comparison with existing approaches shows that the change management and mapping reconciliation process of the proposed system are accurate, consistent, and comprehensive in their coverage.

Thesis Supervisor: Sungyoung Lee
Title: Professor

Acknowledgments

First and foremost, my humble and sincere thanks to the Almighty Allah for showering his blessings upon me. He gave me the strength, courage,and patience during my studies and stay here in Korea. This dissertation represents a great deal of time and effort not only on my part, but also on part of my advisor, Prof. Sungyoung Lee. I am grateful for the time and advice that you provided me over the past three and half years. This thesis owes much of its contents to your ideas and guidance. You helped me in shaping up my research and finding research areas which are theoretically interesting and practically important.

I am also thankful to my thesis committee members for providing insightful and constructive comments to improve the quality of this dissertation. I am especially thankful to Dr. Khalid Latif and Prof. Brian J. d'Auriol. Their constructive criticism on my work and insightful discussions and suggestions helped me in improving this dissertation a lot.

I would also like to thank all the current and former Ubiquitous Lab members for their support and for providing a pleasant working environment. I am very thankful to all my Pakistani friends at Kyung Hee University and especially to Mr. Zeeshan Pervez, Dr. Adil Mehmood Khan, Dr. Mohammad Shoaib Siddiqui, Mr. Ozair Idrees Khan, Mr. Muhammad Fahim, Mr. Wajahat Ali Khan, and Mr. Muhammad Bilal Amin for their time, help, and continuous support.

I have no words to express my sincere gratitude to my parents, sisters, and brothers for their unconditional love, prayers, support, wishes, and encouragement. They have been my strongest motivation to complete this dissertation.

Contents

List of Figures

List of Tables

List of Acronyms

In alphabetical order:

ACID Atomicity, Consistency, Isolation, and Durability

ACM Association for Computing Machinery

CHAO Change and Annotation Ontology

CHL Change History Log

CHO Change History Ontology

CRM Conceptual Reference Model

CVS Concurrent Versions System

DL Description Logic

MSC Mathematics Subject Classification

OMV Ontology Meta Data Vocabulary

OWL Web Ontology Language

OWL-DL Web Ontology Language - Description Logic

OWL-FULL Web Ontology Language - FULL

RDF/N3 Resource Description Logic / Notation 3

rdfs Resource Description Framework Schema

SA Semantic Affinity

SPARQL SPARQL Protocol and RDF Query Language

SWETO Semantic Web Technology Evaluation Ontology

SWRC Semantic Web Research Community

URI Universal Resource Identifier

Chapter 1

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 the representation of domain knowledge [27]. It is essential for: sharing knowledge of research outcomes, sharing information among independent organizations [12], exchange of information among clinics [47], and among heterogeneous systems [11]. To enable such sharing, the need is to model the information and at the same time preserve its semantics. Special attention is required to preserve the semantics while modeling certain aspects of a domain [40]. Ontology provides formal structure (model) with semantics about how an expert perceives the domain of interest with its real meaning. Philosophical ontology is *the science of defining the kinds and structures of objects, properties, events, processes, and relations in every area of reality.* Whereas in computer science domain, *ontology is a formal and explicit specifications of a shared conceptualization of a domain of discourse* [36] and is the main driving force of the Semantic Web vision [10].

The usage of ontology is wide spread in information systems, especially when building a *lingua franca* for resolving the terminological and conceptual incompatibilities be-

tween information networks of varying archetype and different provenance [27, 58, 90]. This in response increases the significance of ontology maintenance [27, 28]. Ontologies are complex in nature and often largely structured. Their development and maintenance incorporates related research areas like: engineering, evolution, versioning, merging, and integration, where these are fundamentally different [27]. For better system accuracy and performance, up-to-date and complete information must be maintained in the knowledge-base. The domain knowledge evolves as the communities of practices concerned with knowledge develop better understanding of their perceived knowledge [93]. Ontology evolution takes place when the perspective under which the domain is viewed has changed [27, 75]. More specifically, ontology evolution means modifying or upgrading the ontology when there is a certain need for change as Communities of Practice concerned with a field of knowledge develop a deeper understanding of the domain. Ontology change management deals with the problem of deciding the modifications to perform in ontology and implementation of these modifications and the management of their effects in dependent data structures, ontologies, services, applications, and agents [26, 27, 58].

1.1 Motivation

Different convergence technologies like: Semantic Web Services [69, 85], Context-aware Search Engines [56], Software Agents [18], Semantic Grid [87], and Cloud Computing [14] use ontologies for their customized needs [6]. Systems using context-aware information (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's customized needs. Especially in combination with mobile devices, this modeled information is of high importance that increases usability of information and applications tremendously [6].

The systems, services, and technology developed and used by independent organizations are autonomous in nature and behave according to organizational needs. The convergence of diverse systems, services, and technologies in use is based on material unity (information modeling and exchange) at every level and of technology integration for that level of unity [87, 61]. Currently in use and most appropriate approach for information modeling, mediation, and integration is the use of ontology in every aspect of data level, system level, service level, and technology level integration and interoperability [6, 40, 61].

The integration and interoperability is maintained not only by maintaining the ontology but there must be a systematic approach that can also handle the dynamic/evolving nature of ontology. Change history management in such evolving knowledge models is a an important and challenging task. 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* [27, 38]. Ontology evolves from one consistent state to another [27, 37] and to accomplish the evolution process several different sub-tasks in a sequence are involved i.e., *Capture change, Change representation, Semantics of change, Change implementation and verification, and Change propagation* [28, 27, 55, 91, 93].

One of the crucial tasks faced by practitioners and researchers in knowledge repre-

sentation area is how to efficiently encode the human knowledge in ontologies? The proper maintenance of these, usually large, structured, dynamic ontologies and in particular adaptation of these ontologies to new knowledge (ontology change) is one of the most challenging problems in the Semantic Web research [28, 31, 58]. Different techniques have been introduced in the research literature to solve the issues of ontology evolution, change history management, and the change effects on dependent data and applications [28, 27, 38, 55, 58, 74, 93].

As an ontology evolves to a new state, the dependant ontologies, applications, and services may become invalid [15, 27, 55, 58]. Consequently, ontology change management solutions must answer a number of questions [32]. One such question is, "how to maintain and manage all the changes in a consistent and coherent manner?" Moreover, the role of change history information becomes critical when an ontology engineer wants to review the ontology evolution history. A storage structure for such information is also crucial for effective retrieval. A single change also makes the existing mappings between ontologies unreliable. In addition, the re-establishment of mappings is also a time consuming process. To handle these challenges, a comprehensive solution is required that must address various multi-faceted issues, such as ontology change history management and traceability, recovery, visualization of change effects, keeping the evolving ontology in a consistent state, mapping reconciliation between evolving ontologies, query reformulation, and collaborative ontology engineering.

1.2 Approaches

There are several approaches to handle the issues of change management and change effects. The existing approaches can be learned/analyzed from three main perspectives as maintenance of changes, change management, and use of changes to eliminate their effects on dependent data, applications, and services.

Relational database management system's techniques are amongst the most popular and mature change management techniques. The changes in relational databases are managed by the database management system for recovery and traceability purposes. The basic purpose of database recovery is twofold. Firstly, it is used to recover the data after system/disk crashes. Secondly, it preserves ACID properties in transactions, and brings the database into a consistent state after transaction errors [23, 50].

Several techniques have been proposed in the literature to maintain database changes, among which logging, check-pointing, shadowing (or shadow paging), and differential tables are the most prominent [23]. Instead of directly updating the actual database tables for a change during a transaction, the intermediate updates are recorded in a sequential file known as transaction log. Database systems can also maintain a checkpoint record in the log. This log can later be used for data recovery, and to bring the database back into a previous consistent state. Similar to the logging method, updates could be accumulated in the differential tables rather than making the changes in the original table or maintaining a complete before image [50]. Three differential tables (i.e., a read-only copy of the base table, a differential table for insertion, and a deletion differential table) are maintained for a single database relation. These different schemes are used for the purpose of change

audit, traceability, and recovery.

Various strategies could be adopted to preserve the changes in ontology, including the use of a database or semi-structured log files. Researchers have provided various techniques to maintain these changes. For example, *Changes Tab* [66] in Protege and *Change Capturing* [81] in NeOn Toolkit, listens to the ontology changes and preserve them in a log file. *Changes Tab* [66] and *Change Capturing* [81] can also be configured for a client-server model in collaborative ontology engineering.

Klein [58] and Flouris [26] have done a significant work on change management for distributed ontologies engineered over a period of time. The author developed change ontology by modeling both the atomic and complex changes. A similar approach is used by D. Liang [63] and [81]. Ontology changes are stored in a file as a script following a temporal ordering. The script follows the specifications provided in the Log Ontology [62] and OWL 2 Change Ontology [81]. Upon the user's request, this script file is used to carry out undo or redo operations. Such log files are maintained for particular editing sessions.

There exist systems like [2, 25, 67, 102, 104] that support mapping evolution which are effected by the the changes in corresponding ontologies. Studies done in [25, 102] mainly focus on schema based mappings evolution to support Local as View and Global as View approaches [61]. This supports query reformulation in data integration applications. The system proposed in [102] focuses on mapping evolution based on incremental adoption of changed mappings. The system discussed in [25] is based on composition and inversion technique. This technique makes the schema evolution restricted to a set

of defined states based on mapping evolution options, which is not true in real world [27].

To support the dynamic nature of the Semantic Web, there must be some mechanism to cope with the continuous evolution of domain ontologies. Therefore, it is important to manage the ontology changes effectively and in a consistent and coherent manner, and to maintain the relationship between the changes and ontology. It is a complicated task while considering the dynamics of ontologies, and is also critical to the support of networked ontologies. Ontology evolution could be amalgamated in a holistic approach to manage ontology change history as well as their effects on dependent services. This approach helps in optimum utilization of resources in terms of computational time and runtime memory usage.

1.3 Problem statement

An ontology evolve from one consistent state to another due to certain changes [27], which on the other hand introduce problems related to management of changes and change effects. Change representation, logging, and propagation (for reconciliation) are the areas that remained partially uncovered. The focus of this research work is to work on ontology change management in general and on change representation, history management, and change reuse for mapping reconciliation in particular (see Figure 1.1). These challenges associated with ontology changes are highlighted in the following subsections.

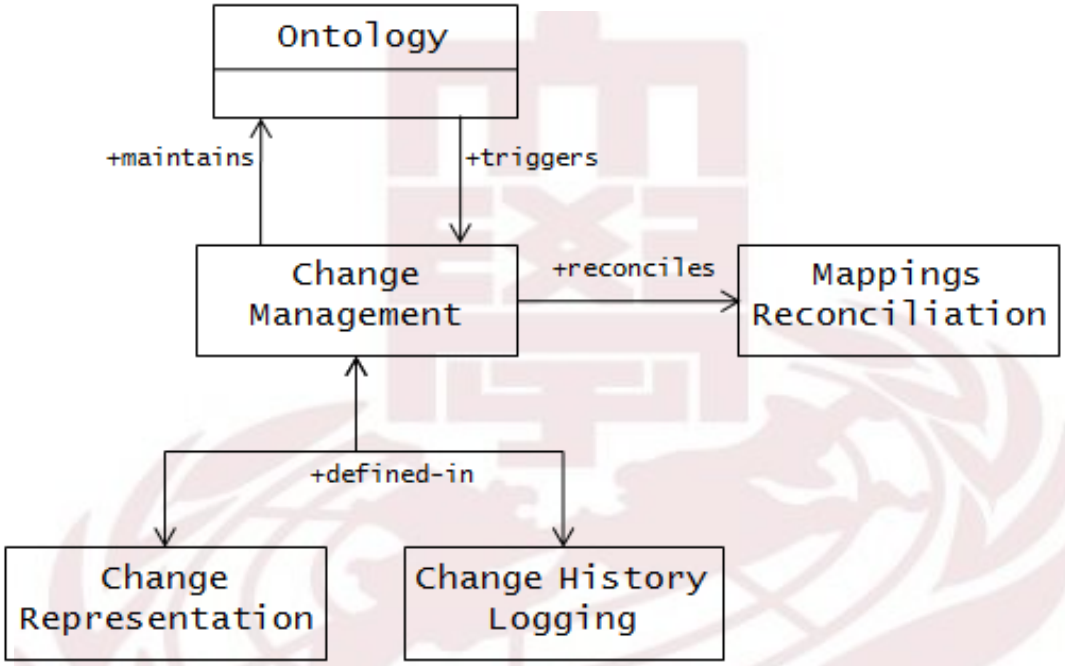


Figure 1.1: Showing the ontology changes, management of change history, and using these changes to reconcile the mapping between ontologies.

1.3.1 Ontology Change Management

Most of the existing systems work with managing ontology changes of two different versions. However, they target ontology versioning rather than ontology evolution. Whereas in ontology evolution, information about the previous state as well as the changes are simply preserved in a sequential log that mix the current information with the existing. The problems that raises and remaind unsolved in ontology change management perspective while managing changes in dynamic ontologies are:

Change Representation

- There is no pragmatic and formal structure for management and representation of changes based on principles of change and standards defined in literature.
- There is no uniform and generic structure that is based on a specific granularity level of changes. This make the existing structure as strictly application specific.

Change Logging

- To refer to any particular state of ontology, storage and access to and associated changes with the referred state is required. However, the current storage structures do not support the associated logging of changes from a particular change session.
- The existing storage structures do not support effective and efficient retrieval of change information for better performance and optimum resource utilization.

Mapping Reconciliation

When an ontology evolves then the changes should also be propagated to its dependent data, applications, and services. However, system that supports change propagation to dependents have following issues.

- The ontology evolution results in making the mappings between ontologies unreliable and stalled. The existing systems fails to re-establish the mappings among the changed ontologies and re-instating the suspended applications in time.
- The existing systems are resource hungry (i.e., it consumes lots of computational time and runtime memory) when considering re-establishment of mappings even for a single change.

In nutshell, there is no comprehensive Change History Management Framework for effective change capturing, formal representation, and logging. And then based on the logged changes, reconcile stalled mappings with efficient utilization of memory and computational resources, while maintain appropriate level of mapping accuracy. Considering the issues associated with ontology changes, the ontology change management becomes more important, not only to manage the changes in a consistent and coherent manner but also to eliminate/minimize the effects of change on dependent applications, systems, and services. In addition, the change history also help to reconcile the mappings between dynamic mapped ontologies and support in reliable information exchange on updated mappings.

1.4 Contributions

Based on discussion in Section 1.2, there exist different techniques to store the change in a data model, for example, check points, shadowing, differential tables, scripts files, and logging. Among these, logging technique has been adopted in this research work to store the changes in an ontology.

The goal of this research work is to provide a methodological framework for ontology change management in general. In particular, the objectives of this research are to: develop a comprehensive storage model for ontology changes, consistent and coherent management of ontology change history, mechanism for ontology change capturing and logging, and reusing the ontology change history for the purpose to reconcile the mappings between evolved (i.e.,changed) ontologies. Figure 1.2 shows the layered approach of the proposed change history management framework towards handling of ontology change and reusing it for mapping reconciliation. All functions proposed are also evaluated for their validity and effectiveness.

The general approach adopted and followed in this research is to achieve the goals of change representation, capturing, maintaining, and managing the ontology changes happened during its evolution in a coherent manner. Moreover, to use the logged changes for reconciling stalled mappings between dynamic ontologies. The subsequent sections presents the main contributions of this research.

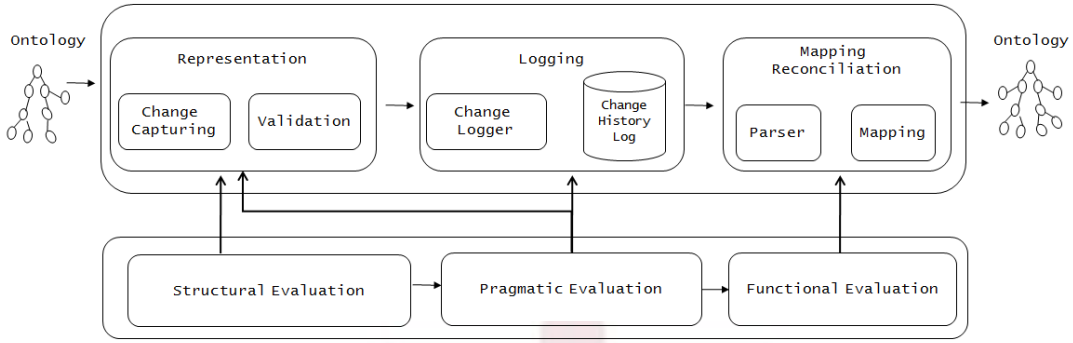


Figure 1.2: Showing the change in ontology handled by the proposed change history management framework including the evaluation criteria.

1.4.1 Change Representation

Ontology is generally used by community of users for knowledge sharing and reuse. When there occurs some changes in the domain ontology then these changes needs to be preserved for later use [27, 58, 81]. To store these changes, mechanisms like databases and text files exist. However, to maintain the changes in a structured and semantically enrich model, a Change History Ontology (CHO) has been developed in this research. To model CHO, a chronological ordering scheme from conditional random fields modeling [103], categorical association scheme from Concurrent Versions System Concurrent Versions System (CVS) and SVN systems [100], and participation patterns of change from ontology design patterns [33] have been used. A comprehensive ontological structure i.e., CHO has been designed that model both element and complex level changes of ontology at atomic level.

1.4.2 Change History Logging

Understanding different ontology changes is necessary to correctly handle explicit and implicit change requirements [27, 38]. For that purpose an ontology has been designed and developed to capture ontology change requirements and keep track of the change history. Similar to relational databases, the proposed methodology relies on a logging technique to persistently store ontology changes. The logged changes in Change History Log (CHL) help in ontology recovery, traceability, query reformulation, and mapping reconciliation. The changes are preserved in a time-indexed manner in a triple store. The change description in CHL conforms to the CHO. Each entry in CHL is modeled as an instance of $OntologyChange$ having association with corresponding changes of that particular change session. The log also preserves the provenance information about the changes, such as who made the changes, and when and why these changes were made to support the change audit.

1.4.3 Mapping Reconciliation

As the domain ontologies are provided by autonomous and independent providers, it makes them evolve independently with flexible structure [39]. The changes result as a change in the already existing mappings and make these mappings unreliable for information exchange. The existing systems consume more time and runtime memory for re-establishing the mappings between ontologies. However, the changes in mapped ontologies and regenerated mediation are not significant [39]. Consequently, less time and memory consuming scheme for reconciling ontology mappings (mapping evolution) is provided using the changes stored in CHL as compared to the existing systems. Proposed

approach uses CHL (i.e., local, centralized, and distributed) for mapping reconciliation to overcome the staleness problem of mappings and produces the results in both time and memory efficient manner.

In short, this dissertation strongly contributes in solving the issue of ontology change history management by designing a structurally sound and semantically enrich ontology structure to capture and maintain the ontology change history. Moreover, the logged changes are reused to achieve the objective of mapping reconciliation with optimum resource utilization.

1.5 Thesis Organization

This dissertation is organized in seven chapters as following.

- **Chapter 1 Introduction**. Chapter 1 is the brief introduction of the research work on ontology change management and in particular on change history management of ontology changes and their utilization. Chapter 1 mainly presents the challenge in focus, the set goals, and the objective achieved in this research work.
- **Chapter 2 Related Work**. Chapter 2 sets the stage by providing the background subject knowledge about ontology evolution and ontology change management. It also compares and contrasts different models, techniques, systems, and related research projects.
- **Chapter 3 Change History Ontology**. A comprehensive description of the design and development of the semantic structure developed for maintaining the ontology

changes is presented in Chapter 3. It also discusses the details of changes and their storage in Change History Log (CHL) using Change History Ontology (CHO). Different examples are also provided for better understanding.

- **Chapter 4 Applications over Change History Log**. In Chapter 4, different potential applications of CHL are discussed with examples and their current status. This chapter also highlights the applications amongst the discussed ones which are developed in this research.

- **Chapter 5 Change Capturing and Mapping Reconciliation**. A comprehensive discussion on the system design and implementation for the change capturing and logging capability is presented in this chapter. This chapter also reuses the logged changes for later use for the purpose of reconciling ontology mappings. Detail procedure for mapping reconciliation is provided in this chapter.

- **Chapter 6 Implementation and Results**. Detail results on change capturing as well as on mapping reconciliation is provided in this chapter. This chapter also presents the proposed scheme's comparison against the existing systems. The proposed scheme's results are discussed in detail from different aspects, such as computational time, runtime memory usage, and accuracy.

- **Chapter 7 Conclusion and Future Directions**. This chapter presents the conclusion of this research work and highlights the main contributions. The future directions for this research work are also mentioned in this chapter, which may need further research efforts and systematic thought.

Chapter 2

Related Work

Ontologies are formal descriptions of a shared conceptualization of a domain of discourse [36]. The main components of ontologies are concepts, properties, individuals, and axioms. Concepts represents classes or entities of a domain, whereas, relations describe the interactions between individuals of these concepts. Individuals are the real world existence of information represented using concepts. To apply constraints on concepts and individuals axioms are being used. Ontologies are developed to capture and model the domain knowledge, and share it with the community of users, machines, systems, and system agents [27, 36]. When there is a change in the domain knowledge then the definitions provided using the domain ontologies will also be changed to reflect the changed knowledge. A formal ontology is flexible and dynamic in nature. It evolve with the evolving domain knowledge in order to keep it consistent with the growing knowledge [4, 27, 58, 105].

A common understanding of a change is that it happens in relation to time. The rate of change and time is different that separate these entities; however, are very much re-

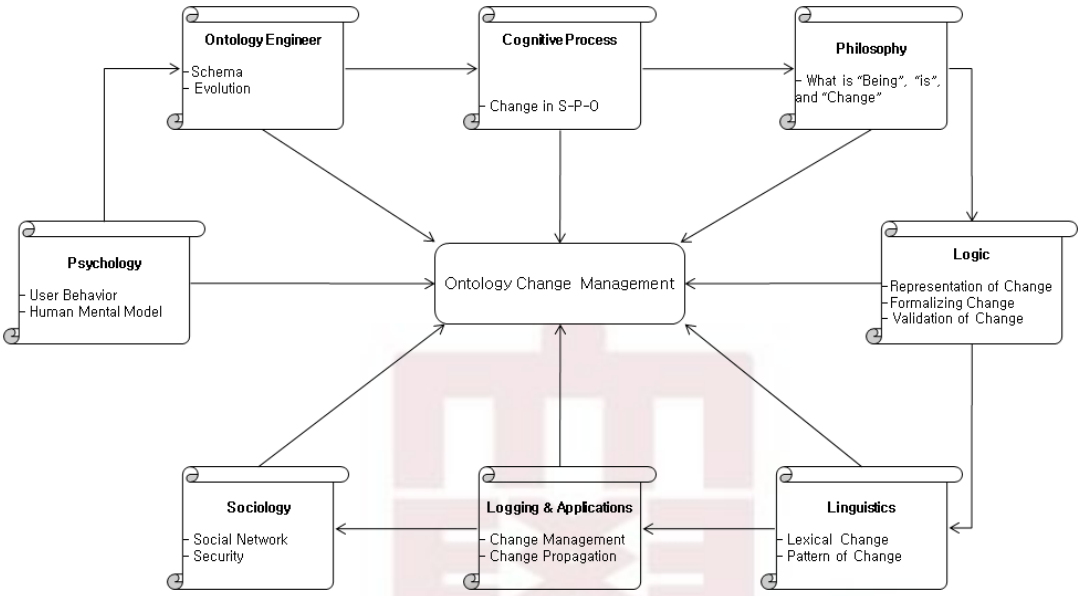


Figure 2.1: Change occurrence and implementation cycle.

lated [27, 42]. The nature of change may vary from small scale to a larger; however, the changes always have its cycle to complete [88]. The changes cycle and the philosophy of changes is depicted in the Figure 2.1. The change will start and passes through different phases and will finally be accommodated in the final model. The final model is in most cases the intermediate model once another change is initiated.

The large and complex structure of ontology raise several interesting challenges relating the development and maintenance of the dynamic ontology. One of such challenges is the ontology change [27, 58]. Several reasons for ontology changes has been identified in the literature. In this dissertation, the term ontology change will be used in a broad sense, covering: changes from external sources/events, changes by the ontology engineer, changes forced by importing another ontology, changes forced by translating the ontology

into a different language or different terminology, changes due to ontology merging and integration [27, 60, 58, 55, 61, 62, 75, 81, 84, 86, 93].

An ontology is just like any other information model holding information regarding a domain of interest. The information model may need to evolve to accommodate new information/change [27, 58, 94]. The changes are new information or updates in existing information, which were previously unknown, or otherwise unavailable, or different features of the knowledge may become important now [27, 44]. Moreover, ontology development is a collaborative process and this process would require the ontology to reach a consistent final state. However, this final state is an intermediate final, as ontology evolution/change process is usually a continuous/ongoing process of ontology modifications [27, 44, 60, 58, 55, 81, 84, 86, 93], (see Figure 2.2 for lifecycle of ontology evolution/change process).

Ontology change management deals with the problem of deciding which modifications to perform in ontology in response to a certain need for change [29]. This mechanism ensures that the required changes are reflected in the ontology, and that it is in a consistent state. It deals with four different aspects [28, 27, 81]. (1) *Ontology evolution* is the process of modifying ontology in response to a certain change in the domain or its conceptualization. (2) *Ontology versioning* is the ability to handle an evolving ontology by creating and managing different versions of it. (3) *Ontology integration* is the process of composing the ontology from information found in two or more ontologies covering related domains. (4) And lastly, *ontology merging* is the process of composing the ontology from information found in two or more ontologies covering highly overlapping or identical domains. The focus of this research work is on the change aspect of ontology

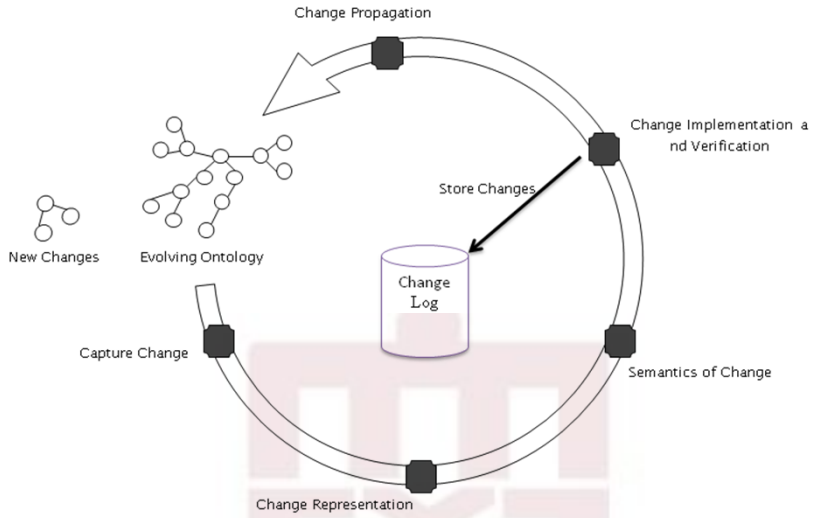


Figure 2.2: Ontology Evolution Lifecycle, takes source ontology along with new changes and implement the new changes to source ontology.

evolution whereas, sometime it is also confused with ontology versioning; however, ontology versioning is a stronger variant of ontology evolution [27, 38, 44, 60, 58, 55, 81]. Moreover, this research is mainly focusing on the schema level of ontology instead of both schema and data level. To understand the difference between schema level and data level, refer to Figure 2.3.

To support the dynamic nature of the Semantic Web, there must be some mechanism to cope with the continuous evolution of domain models and knowledge repositories. Therefore, it is important to manage the ontology changes effectively, and to maintain the relationship between the changes and models [62]. To achieve the objective of ontology evolution, different research have worked on this area. Details of efforts by the researcher is presented in Table 2.1. A lot of research on schema evolution has been carried out

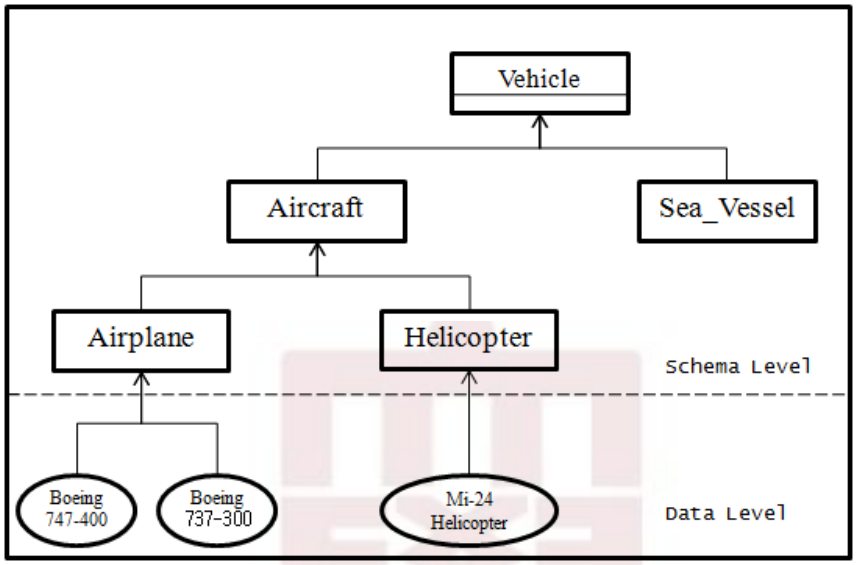


Figure 2.3: The figure shows an example of difference between the schema level and data level. This research is focusing on the ontology changes at schema level.

in relational databases. Schema evolution handles changes in a schema of a populated database without losing data, and provides transparent access to both old and new data through the new schema. It is a complicated task considering the dynamics of ontologies [74], and is also critical to the support of networked ontologies. Ontology evolution and versioning could be amalgamated in a holistic approach to manage ontology changes as well as their effects.

Aproach	Change Request	Change Representation	Conflict Resolution	Change Implementation	Change Propagation	Working
[93]	The complete change request is represented in formal representational format. These changes (due to business requirements) are specified by ontology engineer.		Ontology engineer resolve all the inconsistencies due to requested changes by incorporating deduced changes.	The requested changes (including deduced changes) are applied to the source ontology.	Applied changes are propagated to dependent data, applications, services, and ontologies. Out-of-date instances are simply replaced with the up-to-date instances.	User Dependent
[60], [58], [73]	Specified by ontology engineer.	Change and Annotation Ontology (CHAO) to represent change request.	Ontology engineer involvement.	Suggested that tools should provide interface for user interaction.	Consistent propagation of changes to distributed instances of ontology.	User Dependent
[30]	Specified by ontology engineer	Formal representation of changes	Use predefined strategies for conflict resolution and avoiding side-effects.	Provide interface for user interaction and also log the changes.	Propagation of changes to dependent artefact.	Semi automatic
[83]	Different versions of ontologies are used in this approach. The 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 check for any conflicts.	It does not support change propagation as it works on versions.	User dependent
[80], [86]	Changes are detected among two versions by using PromptDiff and OntoView [59], and compile a complete change request.	Formally represented using their developed semantic structure.	Ontology engineer resolve 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 dependent

Approach	Change Request	Change Representation	Conflict Resolution	Change Implementation	Change Propagation	Working
[68]	Use top-down approach (manual) and bottom-up approach (automatic) for change detection.	Suggestions for use of formal representation i.e., using the log representation for changes.	Involve ontology engineering for resolving conflicts.	Manual implementation of these changes.	It does not support change propagation.	User dependent
[65]	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 is made.	User dependent
[15]	Changes are recognized automatically by analyzing domain artifacts. H-Match [17] and WordNet [71] 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 in focus.	Semi automatic
[55]	New changes such as (change in single concept, group of concepts and concepts in a hierarchical structure) are detected automatically using H-Match [17] and WordNet. Change representation is provided by Change History Ontology (CHO) [52].		For conflict resolution KAON [28] is used with some suggested extensions.	Changes are implemented atomically and after every change implementation, these are logged in CHL [52]. At the end all changes are validated against the change request.	Change propagation is not handled in this approach.	Suggestions toward automation
[106]	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 2^{nd} phase with conflict resolution.	Towards automatic ontology evolution.

Table 2.1: Comparative Review of Ontology Change Management Systems.

2.1 Ontology Evolution Process

The efforts in this section are towards discussion on and identification of deficiencies in current research on ontology evolution, that has extensively been worked in [26, 27, 58, 81]. Viewing ontology as a special type of Knowledgebase gives a different perspective of ontology evolution [27, 74].

The process of ontology evolution has the following two variants: *Ontology Population* and *Ontology Enrichment*. New instances of prior coded concepts can be introduced or existing instances can be updated. As a result, the A-Box is changed and reflects new realities. This is called *Ontology Population*. Where as in *Ontology Enrichment* process, new domain concepts; properties; or restrictions are introduced or existing ones are updated. This later variant refers to the changes in the schema or T-Box. Overall, the process of evolution takes ontology from one consistent state to another [16, 93]. Figure 2.4 depicts an overview of this process and shows an interconnection of needed building blocks. In a holistic manner. These components ensure that the ontology has evolved to a consistent new state, incorporating all the required changes. These components are comprehensively discussed in the subsequent sections whereas the tools supporting the evolution process are discussed in Table 2.2.

2.1.1 Change Detection and Description

The first step in the process is to detect changes, whether the suggested changes are already present in the target ontology. Additionally, schema and individual level differences can be detected effectively, as reported in [98]. In case the concept in focus is totally new

System	Contributions	Limitations	Evolutions
Protege [79, 73]	1. Mostly used for ontology creation 2. Often used for evolution and maintenance 3. Provide Visualization, Merging, Integration, and Comparison 4. SparQL queries support	1. Weak facility for ontology change management 2. No facility for ontology recovery 3. Use third party services for consistency checking of ontology	Manual evolution support
KAON [30]	1. Provide ontology editing services like Protege 2. Provide good environment for pre-evolution strategy making, generate deduce changes to avoid conflicts 3. Support automatic evolution, redo, and undo 4. Provide collaborative editing facility	1. Complex system 2. Slow in response 3. Need ontology engineering for conflict resolution	Pre-defined strategy based evolution support
OilED [7]	1. Used for ontology engineering 2. Disallow inconsistency in ontology 3. Support semi-automated ontology evolution	1. No change logging facility 2. No facility for ontology recovery 3. More strict in its operations compared to Protege	Semi-automatic support
OntoEdit [95]	1. Used for ontology editing 2. Use more options than KAON for strategy making 3. Allow collaborative environment for ontology editing	1. Provide less operations than KAON 2. To avoid side effects of conflicts, it involve ontology engineer	Strategy-based evolution support

Table 2.2: Brief description of ontology editing tools.

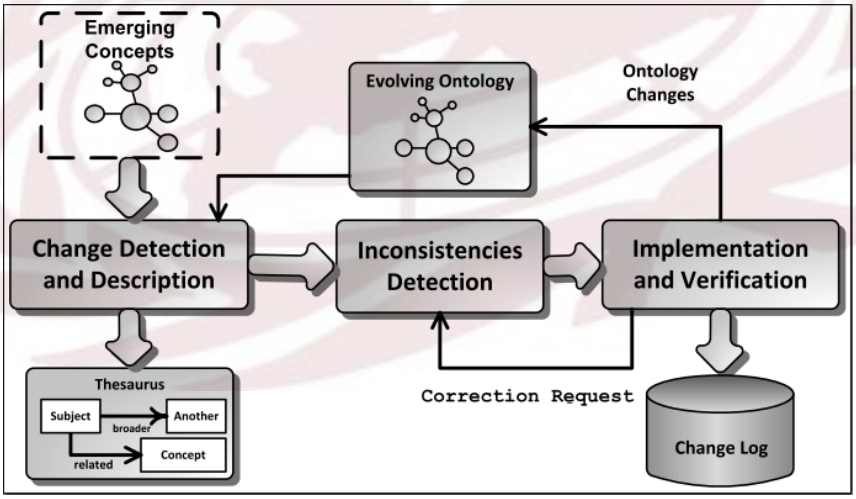


Figure 2.4: The Ontology Evolution Process when a change in ontology is requested.

and there is no additional information, then the H-Match algorithm [17] is used. It takes the new concepts for addition and the target ontology as inputs, and returns the best matching concept in the ontology in order to identify a taxonomic position for the concept [16].

The identified changes: elementary (atomic/simple) change e.g., renaming a class or a property; or composite (complex) change e.g., merging two hierarchies with all their constraints applied, are represented in a consistent format. These changes are first assembled in a sequence, followed by the change implementation. The focus is on atomic changes and also all the composite changes are considered as an ordered sequence of atomic changes. Change History Ontology [52] representation is used to represent changes. This representation is also used to log ontology changes in the Change History Log (CHL) discussed later.

2.1.2 Inconsistencies Detection

In this module ontology changes are analyzed in a systematic manner to ensure that the consistency of the ontology is not lost. The ontology may become inconsistent because of the changes. Two types of inconsistencies can occur: (1) syntactic inconsistencies occur when an undefined or inconsistent construct from meta ontology level is used; (2) semantic inconsistencies occur when the meaning of the ontology entity is changed due to the changes. To keep the ontology in a consistent state, further changes are inferred by taking into account the newly introduced ontology changes, known as induced and deduced changes, respectively.

2.1.3 Change Implementation and Verification

This process covers the following three aspects: 1) change should be applied in complete isolation, and must be atomic, durable, and consistent; 2) each implemented change is verified against the change request; and 3) All the implemented changes must be logged in the CHL to keep track of the changes performed in an ordered manner.

2.2 Change Management

This section introduce different techniques and approaches in ontology domain as well as in the sibling domains for handling the issues of change management.

2.2.1 Database Change Management

It is worth mentioning some of the change management and recovery techniques available in related fields, such as databases. Relational databases also change and these changes are managed by the database management system for recovery and traceability [4, 23, 39]. The basic purpose of database change management is twofold. Firstly, it is used to recover the data after system/disk crashes. Secondly, it preserves (atomicity, consistency, isolation, and durability) Atomicity, Consistency, Isolation, and Durability (ACID) properties in transactions, and guarantees to brings the database into a consistent state after transaction errors.

From the ontology change management standpoint, this research is not strictly con-

cerned with system crashes. This aspect could easily be delegated to the underlying storage system which can maintain a complete backup/shadow of the whole ontology. On the other hand, recovery from inconsistencies doesn't require a complete archival copy of the database. Several techniques have been proposed in the literature to recover databases, among which logging, check-pointing, shadowing (or shadow paging), and differential tables are the most prominent [23]. Instead of directly updating the actual database tables for a change during a transaction, the intermediate updates may be recorded in a sequential file known as transaction log. The log file serves as historical archive of the recent database operations. Database systems can also maintain a checkpoint record in the log. This log can later be used for data recovery, and to bring the database back into a consistent state.

Similar to the logging method, updates could be accumulated in the differential tables rather than making the changes in the original table or maintaining a complete before image [50]. Three differential tables are maintained for a single database relation: 1) a read-only copy of the base table; 2) a differential table for insertion, wherein all the new tuples are first inserted in this differential table; 3) a deletion differential table, wherein all the deleted tuples are maintained in this differential table. Consider, for example, a $Student$ relation. The read-only table for this relation could be referred to as $StudentR$, the insertion differential as $StudentD^+$, and the deletion differential table as $StudentD^-$. The following equation could then be used to reflect the updates in the actual table.

$$Student = (StudentR \cup StudentD^+) - StudentD^- \tag{2.1}$$

2.2.2 Ontology Change Management

Various strategies could be adopted to preserve the changes in ontology, including the use of a database or semi-structured log files. Different researchers have provided various techniques to maintain these changes. For example, *Changes Tab* [66] and *Version Log Generator* [84] in Protege and *Change Capturing* [81] in NeOn Toolkit, listens to the ontology changes and preserve them in a log file. The generated logs are flexible to add, delete, and modify annotations about changes made to the model. *Changes Tab* [66] and *Change Capturing* [81] can be configured for a client-server model in Collaborative Protege and NeOn Toolkit respectively. It also comes with a conventional tabular view for searching and navigating within the changes.

Klein [58] has done a significant work on change management for distributed ontologies. The author developed change ontology by modeling both the atomic and complex changes. A comprehensive categorization of different ontology changes is also provided. In addition, Flouris in [26, 27] provided a detailed classification of ontology changes and the related fields of studies which were very much confused together. A formal procedure for ontology evolution is devised that is based on the concept of belief change tools and techniques. This categorization and the change representation given by [26, 58] are the foundation of this research work. This research has used the basis provided to model a representational structure for ontology changes.

A similar approach to the proposed approach is used by [63] for query reformulation and [81] for collaborative ontology engineering. Ontology changes are stored in a file as a script following a temporal ordering. The script follows the specifications provided

in the Log Ontology [62] and OWL 2 Change Ontology [81]. Upon the user's request, this script file is used to carry out undo or redo operations. Such log files are maintained for particular editing sessions. This way, the command history of one editing session is maintained within Protege and is a candidate for transparent query answering. The *Change Capturing* [81] is responsible for capturing the ontology changes and propagating them to the other instances of the same ontology on different nodes, both locally and remotely [81]. The need is to establish a mechanism for maintaining the changes for a longer time-span to support, for example, mining change patterns in a networked ontologies.

2.3 Ontology Change Tracking

Most of the previously discussed ontology change management systems focus mainly on detection and archiving the ontology changes [58]. These changes are not used; however, for undo/redo operations during collaborative ontology engineering [81], or even for ontology recovery. Change traceability and in particular accurate change detection is still a challenging task [27, 81].

In [86], the authors provided a structure for persistent storage of ontology changes and a limited support for change visualization. The work reported in [86], introduces a scheme of logging and visualizing multiple ontology changes. The process starts with detecting changes in two versions of ontology using Prompt [77] and OntoView [59]. A runtime and offline change detection and tracking techniques have also been presented in [84]. However, during runtime, the technique is totally dependent on the ontology edit-

ing tools. Because, it can only detect changes which are triggered by the editing tools. Whereas in offline approach, some of the changes are missed; moreover, the sequence of changes is also not preserved. A centralized, distributed, and remote change detection as well as management and propagation infrastructure has been presented in [81]. This technique is also configured with NeOn Toolkit to detect changes in ontology at runtime. Then the changes are stored in OWL 2 Change Ontology. These changes are later propagated to dependent applications, ontologies, and services. However, as discussed, this technique also depends on the tool to which it is configured and may miss changes which are committed by the user but not triggered by that editor.

2.4 Ontology Change based Mapping Re-establishment

Recently, ontology is being used by convergent technologies such as; Context-aware Search Engines [56], Software Agents [18], Semantic Grid [87], Cloud Computing [14], and Semantic Web Services [69, 82, 85]. Many research groups are working on ontology matching/mapping and have developed different systems (such as MAFRA [67], Prompt [76], FOAM [22], H-Match [17], Falcon [46], Lily [104], TaxoMap [41], and AgreementMaker [20, 19]) that facilitate interoperability between collaborative convergent technologies.

Mapping systems discussed above are those with outstanding performances at Ontology Alignment Evaluation Initiative. Among the discussed systems, AgreementMaker [20, 19], TaxoMap [41], and Lily [104] are the most efficient and widely used tools for ontology matching and mapping with relatively better accuracy. Also, when alignment is to be

constructed completely from the scratch, their accuracy is better than the other existing algorithms [3, 5, 67, 76, 22, 17, 46]. However, like every other system, AgreementMaker, TaxoMap, and Lily also take considerable amount of time in the establishment of alignments, and they have no support for the process of mapping reconciliation (i.e., to update the stalled/unreliable mappings).

All the above discussed systems re-initiate the process of mapping between ontologies after they are being updated. This consumes lots of time as the changes are usually very simple in type and less in number [27, 39]. There exist systems like [102, 25, 2, 104, 67] that support mapping evolution. However, some of these have different focus while few are not mature enough in their approach.

The systems discussed in [102, 25] mainly focus on schema based mappings evolution to support Local as View and Global as View approaches [61] that supports query reformulation in data integration applications. The system proposed in [102] focus on mapping evolution based on incremental adoption of changed mappings. The incremental adoption technique makes it hard to cope with the drastic schema evolution situation. The system discussed in [25] is based on composition and inversion technique. This technique makes the schema evolution restricted to a set of defined states based on mapping evolution options, which is not true in real world [27].

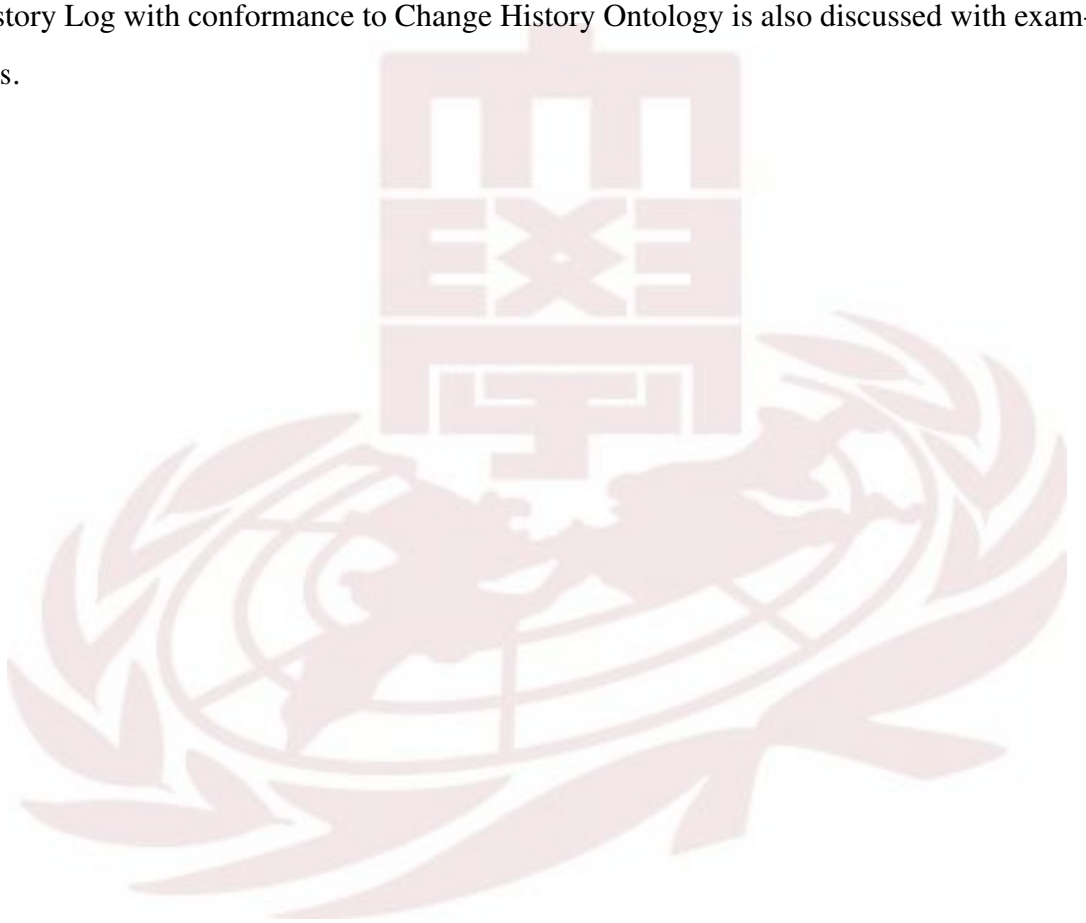
The proposed approach is different from both [102, 25] as schema and ontologies are fundamentally different [2, 74]. In [2], authors proposed a mapping evolution algorithm for mappings between a schema and schema's annotations. The focus of algorithm is to maintain the consistency of mapping between the schemas and their corresponding anno-

tations. Both the systems discussed in [102, 25] and [2] are different from the proposed system as [102, 25] focus on schema level mapping evolution and [2] focus on mapping evolution between schema and annotations (meta data) for the schema.

MAFRA [67] and Lily [104] are the two mapping systems that in addition to mapping generation between two ontology versions also focus on the evolution of mappings when at least one of the mapped ontology evolve from one state to another. However, both MAFRA [67] and Lily [104] have still no concrete methodology to support mapping evolution for evolving ontologies. So for the testing and discussion on proposed system, the authors made extensions to the existing mapping systems to support the mapping reconciliation procedure instead of redeveloping a complete mapping system.

In a nutshell, most of the existing systems discussed here provide a comprehensive ontology change representation schemes which are adoptable for particular applications. Some of the systems do provide the change capturing services; however, limited by the ontology editing tools. The main questions that the existing literature have not answered are: (1) A unified, comprehensive, and generic ontology change representation structure that can maintain the ontology changes independent of any services and applications. In addition, the structure should be based on principles and be useful for all services and applications. (2) The current change capturing and logging techniques are lacking in capturing the overall change information of an ontology as they wholly depends on the ontology editing tools. (3) Lastly, there is no system available to support the mapping reconciliation process. In addition, to minimize the resource (time and space) utilizations in reproducing the mappings and eliminating the staled mappings.

In this chapter, a detailed review of existing literature and systems relating to ontology change management and mapping reconciliation is presented. The sibling domains are also studied for relevant information and conclusions. In next chapter, the details on design and development of Change History Ontology is provided. The notions borrowed from different domains and their use in the design and development is discussed. Change History Log with conformance to Change History Ontology is also discussed with examples.



Chapter 3

CHO: The Change History Ontology

Ontology engineering process consists of sub processes like; ontology modeling, ontology evolution, ontology change management, refining the ontology, ontology matching, ontology merging, ontology integration, and ontology reuse [27, 45, 58]. Due to the dynamic nature of entities in general, and ontology and ontology based applications in particular, the need to have a changes management framework for life cycles of ontology development and maintenance is getting increasingly important.

The goal of this research is to build a change history management framework to trace ontology changes. The framework helps in automatically detecting and logging all the changes triggered by the change request from ontology engineer. The focus is to develop a generic framework for temporal traceability of ontology changes during evolution and reuse the logged changes to minimize the after effects of ontology evolution. To store and log the changes a repository is required to record all the changes in a systematic manner. It can help in providing the facilities of ontology recovery, mapping reconciliation, query reformulation, and collaborative ontology engineering. Whenever required, the logged

changes are accessed and the required operation is performed. It can also provide facility to make the ontology changes temporally traceable by visualizing the change effects on ontology. To provide these facilities, a semantic structure for representing ontology changes is required. The semantic structure to keep track of ontology changes is developed in this chapter. The structure/model is based on a chronological ordering scheme from conditional random fields modeling [103], categorical association scheme from CVS and SVN systems [100], and participation patterns of change from ontology design patterns [33]. The semantic structure and examples of changed information representation are discussed in this chapter one after another.

As discussed in Chapter 2, recovery and change traceability are essential ingredients of ontology change management systems. A semantic structure is proposed for representing ontology changes, referred to as *Change History Ontology (CHO)*. A comprehensive workflow for change traceability is also presented later. The framework helps in automatically detecting and logging all the changes, triggered by the change request from the ontology engineer. Changes can be recorded according to the defined structure in the repository and can later be utilized to minimize the evolution effects. Consequently, this chapter is organized into two major sections: in first section, CHO is introduced with all its basic design and development constructs. This ontology is used to give a defined structure to ontology changes. The second section of this chapter presents the *Change History Log (CHL)* on top of CHO, to store ontology changes. Examples showing the representation of changes in CHL are also presented in this chapter.

3.1 Ontology Change

Change is the only constant and is taking place throughout the passage of time in a given domain. Due to the nature of change, it causes cascading effects, including inconsistencies in domain knowledge. So studying and reasoning about change make sense when things extend through time which means that the temporal changes relating to concept can have different characteristics at different times [101].

From the above discussion, it is clear that with the dimensions like; reason for change, change agent, and actual change; there is another dimension of change i.e., the time dimension [72]. In order to handle the change, ontology engineer need to identify its occurrence, effects, dependency, and time [90]. As mentioned before, the occurrence of change is time independent and can be referred with time. So representing a change to be universally identifiable and independently meaningful, following five aspects must be contained in it that are derived from the discussion above. (1) Who (user) performed the change? (2) When (timestamp) is this change applied? (3) How is this change identified (change identifier) from a change list? (4) What is the change (change element)? (5) Where is the change applied in ontology context (change parameters with respect to ontology)? The first three aspects holds the meta data information of the change, whereas, the fourth and fifth aspects holds the actual information of change items (see Figure 3.1). This change is an example of atomic type change, whereas, to handle these sort of changes, a detailed representational structure is required. The representational structure is discussed in the next section with details for change representation and storage.

The five aspects of change in Figure 3.1 are satisfied in the following manner. (1) User

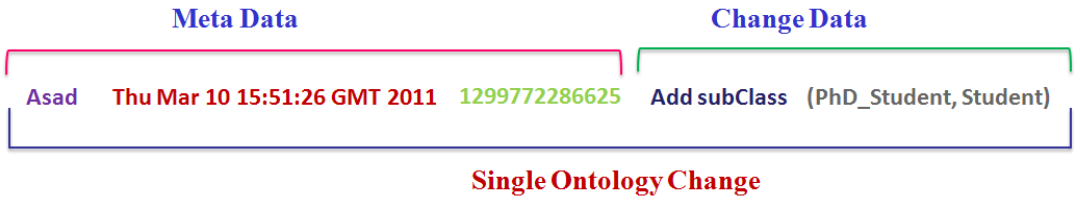


Figure 3.1: Ontology changes including change item with meta data of the changes.

in this example is *Asad*. (2) The timestamp value in example is *Thu Mar 10 15:51:26 GMT 2011* (3) The identifier for this change is *1299772286625* (4) The change operation is *Add subClass*. (5) The change parameters are *PhD_Student* is added as subclass of *Student* class.

3.2 Change History Ontology

A number of changes, ranging from concepts to properties, can affect an evolving ontology. Most of these changes are discussed in greater length in previous literature [8, 58, 81]. The understanding of different ontology changes is necessary to correctly handle explicit and implicit change requirements, classify the changes, and modeling the changes for their unified representation [24, 27, 38, 58, 81]. For that purpose, an ontology to capture ontology change requirements and keep track of the change history have been designed and developed. The proposed *Change History Ontology (CHO)* [52] reuses constructs from existing ontologies [26, 58, 62] and ontology design patterns [33]. New extensions to the existing schemes have been introduced in this research. The notable extensions of CHO are discussed in the subsequent sections.

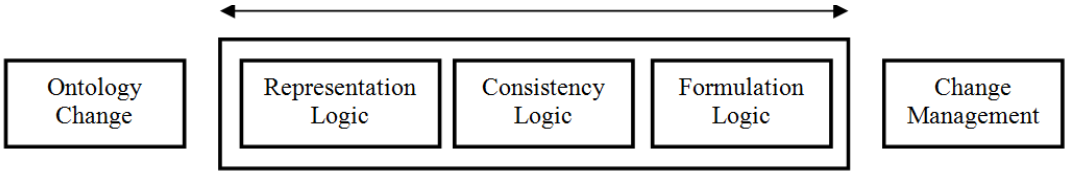


Figure 3.2: Ontology change and its consistency with the change representation scheme and constraints.

Before presenting *CHO*, it is important to highlight some of the main participating characters of it. The core elements of *CHO* are the $OntologyChange$ and $ChangeSet$ classes. The $OntologyChange$ class has a subclass $AtomicChange$ that represents all the class, property, individual, and constraint level changes at atomic level. On the other hand, the $ChangeSet$ bundles all the changes from specific time interval in a coherent manner. The $ChangeSet$ is responsible for managing all the ontology changes and arranges them in time indexed fashion. This time indexing also classifies the $ChangeSet$ as $Instant$ type and $Interval$ type. $Instant$ type $ChangeSet$ holds only one change occurred at some time instant, whereas the $Interval$ type $ChangeSet$ holds the changes occurred in a stretched time interval.

Different resources are referred using different prefixes in this work. It is important to explain the prefixes prior to their use in this research work. The concepts under the "ch" prefix are used from change history ontology and the prefixes "log" and "bib" represent the repository log and an example of evolving *Bibliography ontology* respectively. "xml" is used for xml, "rdf" for rdf, "rdfs" for rdfs, and "owl" for owl namespaces.

3.2.1 Change Handling

The CHO is strictly bounded with the principles of change in ontology/knowledgebase. To satisfy the validity of a change, a change must have three basic properties i.e., *Minimal*, *Success*, and *Validity* [1] discussed below. To validate the change, the change must go through the connected process of validatory (see Figure 3.2). The execution of this process depends on change validity constraints (for consistency, discussed in this section) and change formulation procedures introduced in later sections of this chapter.

1 The *Principle of Minimal Change* enforce that the modifications supposed to be applied on ontology should be minimal. The applied changes are kept at atomic or minimal level for the purpose to avoid any drastic change in ontology. To enforce the minimality of change, the concept of transaction ACID (atomicity, consistency, isolation, and durability) property is applied [34]. The ACID property axioms are given below, whereas, for the given case database transaction is considered as a change and database is considered as ontology. The below given axiom enforce that at particular time, there should be only one change which satisfy the atomicity property of a change.

$Change \equiv \exists X\{X|X \in \Delta, X.resources.lock = exclusive\}$

To add in the atomicity of change, executing it in isolation is implemented using below axiom.

$Change \equiv \exists X\{X|X \in \Delta, X = 1\}$

After having all the resources rights and executing in isolation, the next constraint to be verified before reflecting the change results to dependent ontology and applications is consistency. The constraints are to verify the consistency of ontology

and once the change is executed then the results will completely be reflected to the dependent ontology and application. The axioms for consistency of ontology after change is given in the Principle of Validity, whereas, durability of change implementation on ontology is enforced in the below given axiom.

$Ontology \equiv \{\{Ontology{-}Change\} \sqcup \{Ontology{+}Change\}\} \sqcap \{Ontology.consistent = true\}$

Based on the above discussion and propositions, below given axiom is used to enforce the overall minimality of change. It represents the notion (constraint) for keeping the change at minimal level.

$Change \equiv \exists X\{X | X \in \Delta, X = 1 targetChange\}$

2 The *Principle of Success* observe and satisfy the priorities of alternate changes. Mostly, there are alternate changes available for a given change request. So before a change is applied, the set of changes (alternate changes) are tested for their final result. A change with minimal effects and complete execution is selected for the final implementation. The axiom given below satisfy this principle implementation.

$Change \equiv \exists X\{X | X \in \Delta, \Delta = \{C_1, ...C_n\}\} where C = ChangeInstance$

$Change = 1 targetChange \sqcap min.(Change.Effects)$

3 The *Principle of Validity* enforces that when a change is applied then the ontology must evolve to a new consistent state. Any change that cannot satisfy the consistency constraint is not applied to an ontology. The ACID property consistency is formulated and applied.

$Change \equiv \exists X\{X | X \in \Delta, Ontology.consistent(change) = true\}$

This principle of validity is enforced using the following axiom based on the consistency validation borrowed from the ACID property of transactions.

$$Change \equiv \exists X\{X|X \in \Delta, \Delta = \{C_1, ...C_n\}\} where C = ChangeInstance$$

$$Change = 1 targetChange \sqcap Ontology.consistent.(Change) = true$$

To enforce the change implementation principles on *CHO*, *CHO* has been modeled to capture the changes at atomic level and all the changes must be applied in isolation. In addition, the change should completely satisfy the ACID property of a transaction. A change when implemented is atomic, either it should be performed completely or should not be performed at all. No other change is handled until the current change has been processed and stored. This make the change to be handled in isolation and results in a consistent evolved state of an ontology. After the formulation/handling of change, the change is formally logged in *CHL* for future use that facilitate the durability of change handling.

3.2.2 Change Set

The notion of $ChangeSet$ was introduced a couple of years ago [52]. The same has also been suggested in Change Set Vocabulary [35, 97]. The rationale is that individual changes are not performed in isolation and are usually part of a particular session. On the contrary, $ChangeSet$ can be used to group the individual changes from a particular session in order to incorporate a holistic view over an ontology evolution. Logging changes in sets also helps in maintaining and managing the ontology changes corresponding to specific sessions which is also required for mapping reconciliation, as discussed later. The use of $ChangeSet(s)$ is common in versioning systems, such as CVS and SVN. A

$ChangeSet$ holds information about the changes made during an ontology engineering session. A $ChangeSet$ can span over a stretched interval of time. Its members, atomic changes, are singleton changes on an ontology element at some instance of time. Different changes can be part of the $ChangeSet$, such as modifying details of an ontology class or adding a new object property. $ChangeSet$s also help in maintaining the sequence and grouping of changes. The example in Figure 3.3 of $ChangeSet$ instance covers an ontology editing session spanning over half an hour.

```
log:ChangeSet01
    rdfs:type             ch:ChangeSet ;
    ch:hasChangeSetType   log:Interval ;
    ch:hasChangeAuthor    log:ChangeAgent01 ;
    ch:startTime          "2010-01-01 15:12:58+1" ;
    ch:endTime            "2010-01-01 15:43:11+1" ;
    ch:hasChangeReason    "Concept X is split into two levels" ;
    ch:targetOntology     http://seecs.nust.edu.pk/vocab/bib .
```

Figure 3.3: A $ChangeSet$ example with corresponding meta data including the change agent information, the reason for change, the changed ontology, and the start and end times of the change event. The Figure shows the time spam of $ChangeSet$ as well.

3.2.3 Provenance

The proposed change history ontology is also designed to captures provenance information, such as the change author, reason, and timestamp of change. The author can be an ontology engineer making changes using an ontology editor, or a software agent requesting for some changes, such as an agent during an automatic ontology mapping task. Figure 3.4 depicts an instance of the $ChangeAgent$ class from ***CHO***.

```
log:ChangeSet01
    rdfs:type            ch:ChangeSet ;
    ch:hasChangeSetType  log:Interval ;
    ch:hasChangeAuthor   log:ChangeAgent01 ;
    ...
log:ChangeAgent01
   rdfs:type             foaf:Person, ch:ChangeAgent;
   ch:fullName           "Asad Masood".
```

Figure 3.4: Representation of $ChangeSet$ author information using CHO.

3.2.4 Change Types

The change history ontology supports three types of change operations corresponding to the CRUD interfaces in databases (except the read operation). ***Create*** allows the addition of new facts and vocabulary in ontology, such as $ClassAddition$, $PropertyAddition$, and $IndividualAddition$. ***Update*** operation is used for modifying existing triples, such as renaming a class, property, and individual through $ClassRename$, $PropertyRename$, and $IndividualRename$ respectively. And lastly, ***Delete*** operation serves for removing axioms from the ontology, such as $ClassDeletion$, $PropertyDeletion$, and $IndividualDeletion$. The following axioms depict parts of the conceptual representation of this aspect.

$$\begin{aligned} OntologyChange &\equiv \exists changeTarget.(Class \sqcup Property \sqcup \\ &\quad Individual \sqcup Ontology) \sqcap \exists changeType.(\\ &\quad Create \sqcup Update \sqcup Delete) \end{aligned} \tag{3.1}$$

$$ClassChange \equiv OntologyChange \sqcap \forall changeTarget.Class \tag{3.2}$$

$$ClassAddition \equiv ClassChange \sqcap \forall changeType.Create \tag{3.3}$$

$$\begin{aligned} SubClassAddition &\equiv ClassAddition \sqcap \forall targetSubClass.Class \\ &\quad \sqcap = 1 targetParent \end{aligned} \tag{3.4}$$

For instance, the snippet in Figure 3.5 represents the addition of a new subclass. $SubClassAddition$ is defined as a subclass of $ClassAddition$ and is a type of $ClassChange$ event. The $hasChangedTarget$ represents the newly added class in the *Bibliography ontology*. As the new class is a subclass, the $hasTargetParent$ property connects the newly added class with its parent class through a subclass assertion. The $hasTimeStamp$ represents the exact time of the change event, whereas, the $isPartOf$ connects the change to the corresponding $ChangeSet$ instance.

In the below points, some of the modifications that are made to *CHO* during implementation of the system are discussed. These changes are then reflected in the final version of *CHO*.

- The first version the *CHO* was modeled in Web Ontology Language - Description Logic (OWL-DL) and all changes related to classes and properties were represented

```
log:ClassAddition01
    rdfs:type            ch:SubClassAddition ;
    ch:hasChangedTarget  bib:Transaction ;
    ch:hasTargetParent   bib:Journal ;
    ch:hasTimeStamp      "2010-01-01 15:12:59+1" ;
    ch:isPartOf          log:ChangeSet01 .
```

Figure 3.5: An example of Transaction class addition as a subclass of parent class Journal and its representation using *CHO*.

using a datatype property $hasChangedTarget$ with a string value. Consequently, it required string manipulation to process the entries in/from *CHL*, which is error prone and may lead to incorrect results. To overcome this problem, the *CHO* is transformed from OWL-DL to Web Ontology Language - FULL (OWL-FULL) and $hasChangedTarget$ is declared an object property and its range is set to owl:Class. A similar treatment is applied to $isSubClassOf$, $isSubPropertyOf$, $hasDomain$, $hasRange$, $isInverseOf$.

- The datatype property $hasChangeSetType$ has also been changed to an object type property and created a new class $ChangeSetType$ having two instances as $Instant$ and $Interval$. Now the property $hasChangeSetType$ connect the class $ChangeSet$ with $ChangeSetType$ for its type value.

- Each $ChangeSet$ also needs to maintain information about the ontology to which its corresponding changes are made. In previous version a datatype property $hasOntology$ having string value was used; however, during implementation the property has been changed to object type property and its range is set as owl:Ontology.

- Two other properties $isSubClassOf$ and $isSubPropertyOf$ are also changed to object type properties and their ranges are set to owl:Class and rdfs:Property re-

spectively. The reason for their conversion is to log complete Universal Resource Identifier (URI) of the resources they are referring to.

3.2.5 Temporal Ordering

A time stamp is added with each ontology change. Though a single change is performed at an instance of time, it is common that several changes are performed over an extended time interval. A single change is modeled as a change at a time instance, whereas a sequence of changes is considered as one $ChangeSet$ spanned over a time interval. So for every change entry that corresponds to a $ChangeSet$, a $timestamp$ value is added. This helps in keeping the ontology change entries in an order. The snippet in Figure 3.6 is an example of a $timestamp$ value added to a $PropertyDeletion$ event.

```
log:PropertyDeletion01
    rdfs:type           ch:PropertyDeletion ;
    ch:hasChangedTarget bib:title ;
    ch:hasPropertyType  owl:DataType Property
    ch:hasTimeStamp     "2010-01-01 15:24:31+1" ;
    ch:isPartOf         log:ChangeSet01 .
```

Figure 3.6: Example of showing the $timestamp$ value attached with the $PropertyDeletion$ change instance using *CHO*.

3.2.6 Conceptual Design Patterns

Recently, different ontology development methodologies have emerged [9, 49, 99], some of which advocate the reuse of concepts and patterns from foundational ontologies [33].

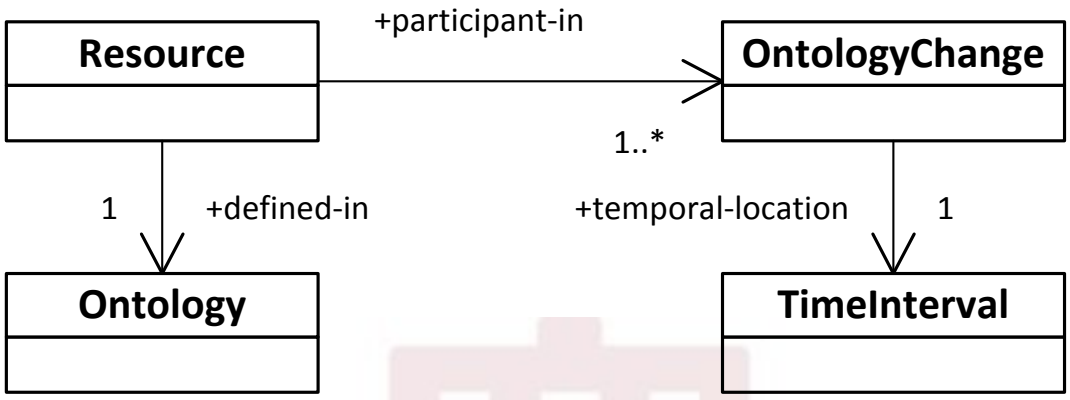


Figure 3.7: Realizing Participation Pattern in Change History Ontology.

More specifically, patterns are useful in order to acquire, develop, and refine the ontologies. Two of the fundamental ontology design patterns have been reused in this research. The *Participation Pattern* consists of a $participant\_in$ relation between the ontology resource and the change event, and assumes a time indexing for it [33]. Time indexing is provided by the temporal location of the change in a time interval, while the respective location within the ontology space is provided by the participating objects (see Figure 3.7 and 3.9). As an example, consider Figure 3.8 as the description of a $ChangeSet$.

In Figure 3.8, a $ChangeSet$ instance is described using *CHO*. The start and end times of the changes are reflected by $startTime$ and $endTime$, respectively. It also logs information about the change agent and the reason for the changes.

CHO is the backbone of the proposed framework. It binds different components of the framework togethers in order to effectively recover ontology from its previous state. In most of the previous approaches, ontology changes are stored sequentially without

```
log:ChangeSet192
    rdfs:type            ch:ChangeSet ;
    ch:hasChangeSetType  log:Interval
    ch:hasChangeAuthor   log:ChangeAgent2 ;
    ch:startTime         "2010-01-02 16:32:58+1" ;
    ch:endTime           "2010-01-02 16:53:11+1" ;
    ch:hasChangeReason   "Changes after applying
                          rigidity meta property." ;
    ch:targetOntology    http://seecs.nust.edu.pk/vocab/bib .
```

Figure 3.8: Example of a $ChangeSet$ instance spanning over a time interval.

preserving their dependence or interlinking with other changes [26, 27, 35, 58, 62, 81]. *CHO* on the contrary, uses $ChangeSets$ to group and time index the changes in a session to preserve coherence of all the ontology changes. A $ChangeSet$ is a setting for atomic changes. One ontology resource participates in a change event at one time interval. Figure 3.9 shows diagrammatic depiction of this pattern. The complete change history ontology is available online[1]. Core classes and concepts in the ontology are also shown in the Figure 3.10.

3.2.7 *CHO* Modeling Language

To represent the intricacy of changes in, classes, properties, individuals, and constraints; quite a large number of classes with associated object and datatype properties are modeled. This helps in recording all the relevant information about a specific change. The properties in *CHO*, which link the change with its target, are represented as annotation properties. Similarly, object properties are modeled to hold the information of changing

[1]http://uclab.khu.ac.kr/ext/asad/CHOntology.owl

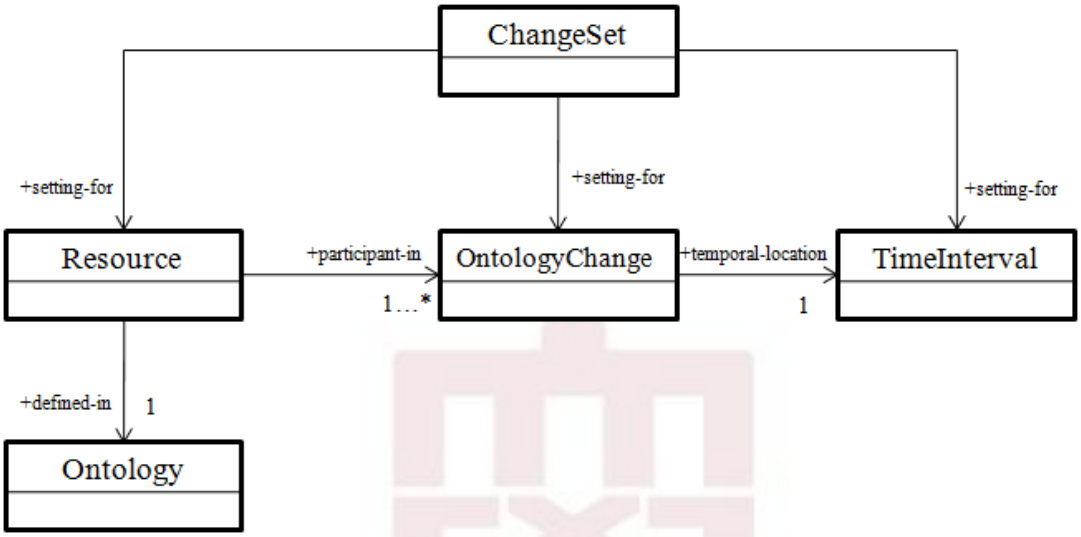


Figure 3.9: Reification of time-indexed participation: ChangeSet is a setting for a change event, ontology resources participating in that change event, and the time interval in which the change occurs.

classes by setting their range to *owl:class*. Consequently, the model still conforms to OWL-DL; however, it supports very limited Description Logic (DL) inference. Advantages of this approach are: reducing the likelihood of error, avoiding string manipulation, and removing ambiguity about the change target.

$$SubClassAddition \equiv ClassAddition \sqcap \forall targetSubClass.Class$$
$$\sqcap = 1 targetParent$$

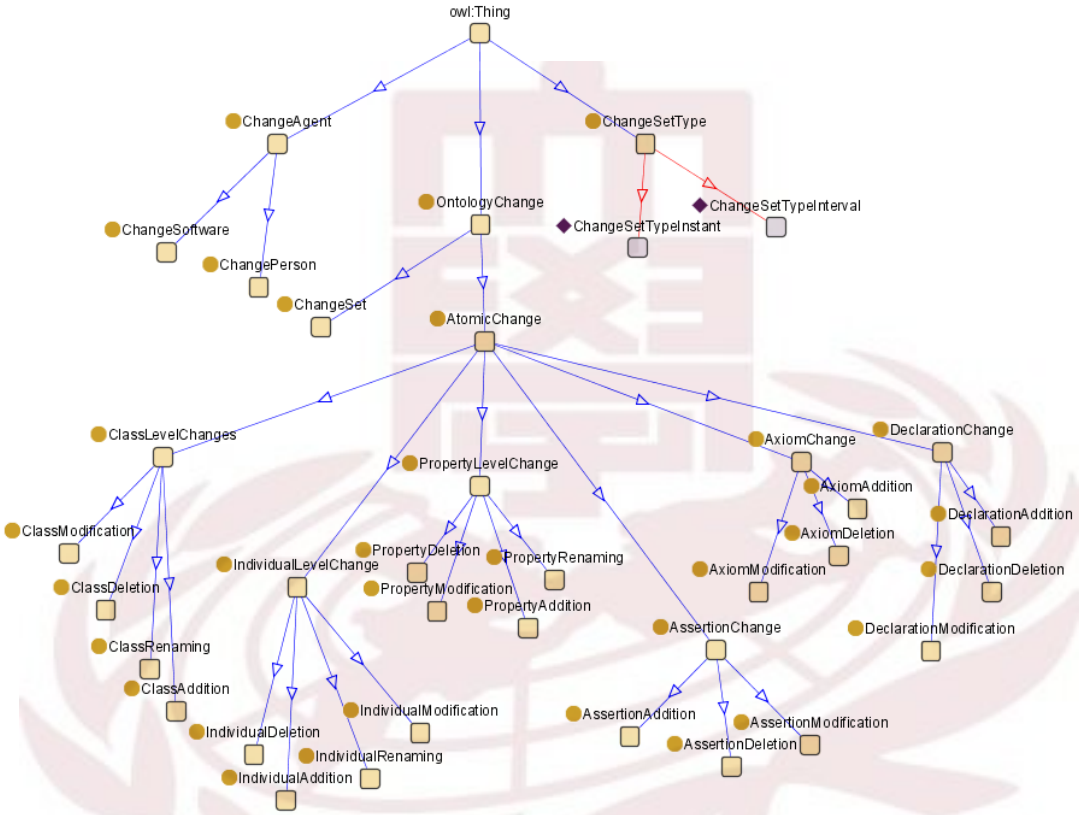


Figure 3.10: Snapshot representing core classes of Change History Ontology.

3.2.8 Complex Changes

In addition to simple class additions, deletions, and renaming's; complex facet modification information is also recorded. Examples include property scope restrictions, equivalence, disjointness, and complex union classes. Similarly, property modification details, such as change in domain/range, setting upper-bound and lower-bound for property values, symmetric, equivalent, inverse, and functional property axioms are also recorded.

Consider an ontology given in Figure 3.11, the *Document* class has two subclasses: *TechnicalDocument* and *ResearchDocument*. The *Document* class is also the domain of an object property *author*. Suppose, due to some reasons the *Document* class gets deleted. This deletion is a complex change event as it will also result in deletion of the subclasses and also unsetting the domain of *author* property. Proposed system records all the changes one by one at atomic level in a sequence. In this deletion event, the change order triggered by Protege is the deletion of the subclasses first, then the deletion of domain of a property, and at the end the deletion of the *Document* class. Proposed system listens to all these changes and logs them in *CHL*. Figure 3.12 represent information about changes made to the ontology and represented in *CHL* as a sequence of atomic changes resulting from a complex change event.

3.3 Change History Log (CHL)

In this section, the change history logging scheme has been introduced. On top of *CHL*, different applications are possible and are briefly discussed in next chapter. The subsequent chapter introduce and explain different applications (CHL-based applications).

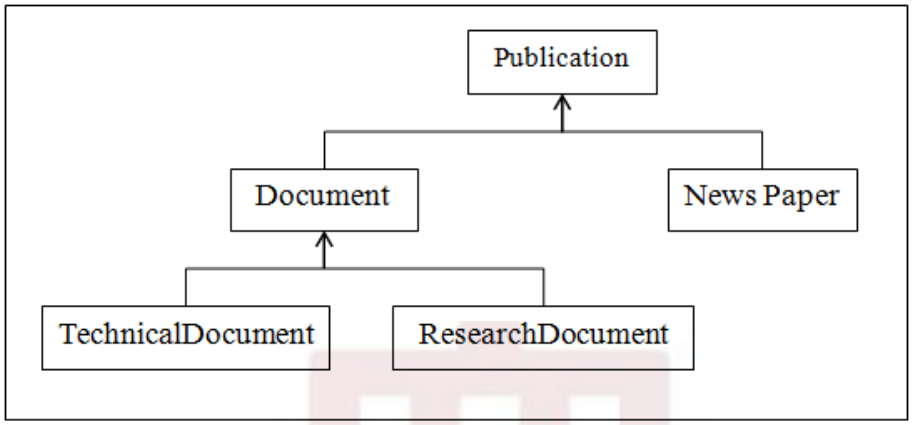


Figure 3.11: An example ontology showing the complex change scenario.

Similar to relational databases, the proposed methodology relies on a logging technique to persistently store ontology changes. Logged changes help in recovering a previous state of ontology after, for example, un-authorized changes, version conflicts, or even an inconsistent state of ontology due to accidentally closing the ontology editor. The changes are automatically preserved in a time-indexed manner in a triple store embedded with the framework. Recovery is manually triggered by a knowledge engineer collaboratively building the ontology. The change description in the log conforms to the *CHO*. Each entry in the log is an instance of either the $ChangeSet$ or $OntologyChange$ class. The log also preserves the provenance information about the changes, such as who made the changes, and when and why these changes were made.

The proposed change history management framework offers numerous benefits ranging from reconciling ontology mappings to increased understanding of ontology evolution process. A running example featuring a *Bibliography* ontology to show the process of log-

```
log:ClassDeletion01
    rdfs:type            ch:SubClassDeletion ;
    ch:hasChangedTarget  bib:TechnicalDocument ;
    ch:hasTargetParent   bib:Document ;
    ch:hasTimeStamp      "2010-01-01 15:14:47+1" ;
    ch:isPartOf          log:ChangeSet01 .

log:ClassDeletion02
    rdfs:type            ch:SubClassDeletion ;
    ch:hasChangedTarget  bib:ResearchDocument ;
    ch:hasTargetParent   bib:Document ;
    ch:hasTimeStamp      "2010-01-01 15:14:47+1" ;
    ch:isPartOf          log:ChangeSet01 .

log:DomainDeletion01
    rdfs:type            ch:DomainDeletion ;
    ch:hasChangedTarget  bib:author ;
    ch:hasDomain         bib:Document ;
    ch:hasPropertyType   owl:ObjectProperty ;
    ch:hasTimeStamp      "2010-01-01 15:24:48+1" ;
    ch:isPartOf          log:ChangeSet01 .

log:ClassDeletion03
    rdfs:type            ch:ClassDeletion ;
    ch:hasChangedTarget  bib:Document ;
    ch:hasTimeStamp      "2010-01-01 15:14:48+1" ;
    ch:isPartOf          log:ChangeSet01 .
```

Figure 3.12: Complex (compound) change resulting from a single change event (Document class deletion).

ging the changes in *CHL*. Consider two changes as a part of one $ChangeSet$. First change adds a new $Author$ class in the ontology and the second change sets $Author$ as $range$ of the property $hasAuthor$. Firstly, the process of creating entries in the log is explained. Taking into account the first change, an individual of $ClassAddition$ is instantiated. The $isPartOf$ property of this change instance is set to the active $ChangeSet$. Secondly, the $hasTimeStamp$ value of the atomic change is also recorded for time-indexing of change entries. For logging the range addition entry, an individual of $RangeAddition$ class from change history ontology is created and the value for its $hasChangedTarget$ predicate is set to the *Object property* for which the range has changed. The modified range information is then stored as a value of $hasTargetRange$ property. Like all log entries, the $isPartOf$ property of the individual is set to the active $ChangeSet$ and its $hasTimeStamp$ value is also stored with the individual. The code snippet given in Figure3.13 represents these changes in Resource Description Logic / Notation 3 (RDF/N3) format.

In this chapter, a semantically enrich structure (i.e., Change History Ontology) for representing the ontology changes with major constructs like $ChangeAgent$, $ChangeSet$, and $OntologyChange$ is developed. Their working details and logging information in Change History Log with $timestamp$ are all elaborated with examples. As the changes are all logged in the *CHL*, which are usable for different applications. The potential applications of *CHL* are discussed in next chapter with suggested implementation procedures.

```
log:ChangeSet192
    rdfs:type            ch:ChangeSet ;
    ch:hasChangeSetType  ch:Interval
    ch:hasChangeAuthor   log:ChangeAgent2 ;
    ch:startTime         "2010-01-02 16:32:58+1" ;
    ch:endTime           "2010-01-02 16:53:11+1" ;
    ch:hasChangeReason   "Changes after applying
                          rigidity meta property." ;
    ch:targetOntology    http://seecs.nust.edu.pk/vocab/bib .

log:ChangeAgent192
    rdfs:type            ch:ChangeAgent, foaf:Person ;
    foaf:name            "Administrator" .

log:IntervalChangeSet2457
    rdfs:type            ch:ChangeSet ;
    ch:hasChangeAuthor   log:ChangeAgent192 ;
    ch:startTime         00:00:46 ;
    ch:endTime           00:03:21 ;
    ch:hasChangeReason   "User Request" ;
    ch:targetOntology    http://seecs.nust.edu.pk/vocab/bib .

log:ClassAddition245701
    rdfs:type            ch:ClassAddition ;
    ch:hasChangedTarget  bib:Author ;
    ch:hasTimeStamp      1224702057078 ;
    ch:isPartOf          log:ChangeSet192 .

log:RangeAddition245701
    rdfs:type            ch:RangeAddition ;
    ch:hasChangedTarget  bib:hasAuthor ;
    ch:hasPropertyType   owl:ObjectProperty ;
    ch:hasTargetRange    bib:Author ;
    ch:hasTimeStamp      1224702072640 ;
    ch:isPartOf          log:ChangeSet192 .

log:ClassRenaming245734
    rdfs:type            ch:ClassRenaming ;
    ch:hasChangedName    bib:TechnicalDocuments ;
    ch:hasOldName        bib:Technical_Documents ;
    ch:hasTimeStamp      1224702057078 ;
    ch:isPartOf          log:ChangeSet192 .
```

Figure 3.13: Example of changes in *Bibliography ontology* represented using change history ontology constructs.

Chapter 4

Applications of Change History Log

The proposed change history management framework offers numerous benefits ranging from reconciling ontology mappings to increased understanding of ontology evolution process. Ontology evolution propagate side effects of ontology evolution on the dependent data, applications, systems, system agents, services, and other ontologies. For all the challenges/issues/side effects discussed in different scenarios in subsequent sections, possible solutions are suggested to overcome these challenges or even to minimize their effects. In discussion on these challenges, the problems at class level (concept) are presented in the subsequent sections; however, it is applicable to all including slots, instances, and restrictions.

In previous chapter, the change history logging scheme has been introduced. On top of *CHL*, different applications are possible and are discussed in details with potential suggested solutions in this chapter. The subsequent sections explains these applications in detail.

4.1 Ontology Recovery

Changes in an ontology makes the ontology to change from one state to another state. Proper maintenance of the changes is very important to provide the facility of reverting back these changes on the ontology to get the previous consistent state of ontology. These stored changes not only provide the facility for rollback, but is also used for roll-forward operations based on user request. Different ontology editors like Protege [79], KAON [30], and OntoEdit [95] 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 as discussed in Chapter 2, Table 2.2.

For the recovery of ontology, the proposed change history framework is implemented as an enabling component for the ontology editor (i.e., Protege) [51]. It is designed to be implemented as a plug-in for different ontology editors provided they support the hooks implemented in the developed 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, please refer to [51]. The recovery process is still not mature as the proposed recovery is only valid for structure level recovery of ontology; however, there is still no system available that can work for both structure and instance level recovery that will also guarantee smooth operations of web service using this ontology.

Change history log records all the changes after time-indexing them as per the design pattern in the *CHO*. Time-indexing helps in recovering the ontology into a previous consistent state [53]. Managing ontology changes during evolution is also helpful for a

new user to get understanding of the changes made. In addition to *change reason*, annotations can also be added with all the logged changes and associated artifacts to help understanding the changes in ontology, data, and application [52]. The log can also be used to understand the semantics of change on the available ontology constructs.

4.2 Visualization

The Visualization module is responsible for visualizing the ontology, ontology changes, and their effect on ontology. This visualization is in graph like structure rather then tree like structure, because the ontology with class and sub-class hierarchy can also have associative relationships with other classes [58]. Figure 4.1 is an interface for visualizing ontology and visually navigating through its different states. Ontology components (such as concepts and their relationships) as well as the changes made in the ontology are visualized. Effects of these changes, for example, how it evolved to the current state, are emitted by navigating through the life of ontology.

In order to visualize changes, the ontology change parser processes the requested $ChangeSets$ and their corresponding changes. The changes are reverted or implemented on the ontology with the help of recovery module to take ontology to a previous or next state. The *TouchGraph* API is extended for graph drawing in order to visualize the graph view of the ontology structure. Resources, such as classes, are depicted as nodes. These nodes are connected through properties, which are depicted as edges. The direction of an edge depicts the direction of the relationship among the nodes.

Number of filters are supported in the graph view, such as zooming in and out of the graph and fish-eye view. A modified version of the *Spring* graph drawing algorithm [64] is implemented in the visualization that ensures aesthetically good looking graph structure and well separated nodes. The playback and play-forward features where not only the ontology but the changes can also be navigated are provided using the proposed scheme. The visual navigation of changes and change effects on ontology helps in analyzing the trends (Figure 4.1). Starting from the very first version of the ontology, the user can play the ontology changes and their effects on ontology and resources. The changing concepts are highlighted and color coded to reflect the changes. For example, the deleted concepts fade out and the new additions gradually appear in the graph. This improves understanding of the evolution history of ontology.

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. Such a visualization of change effects on ontology can help in temporally tracing the ontology changes and better understanding the evolution behavior of ontology [53]. When a user requests 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.

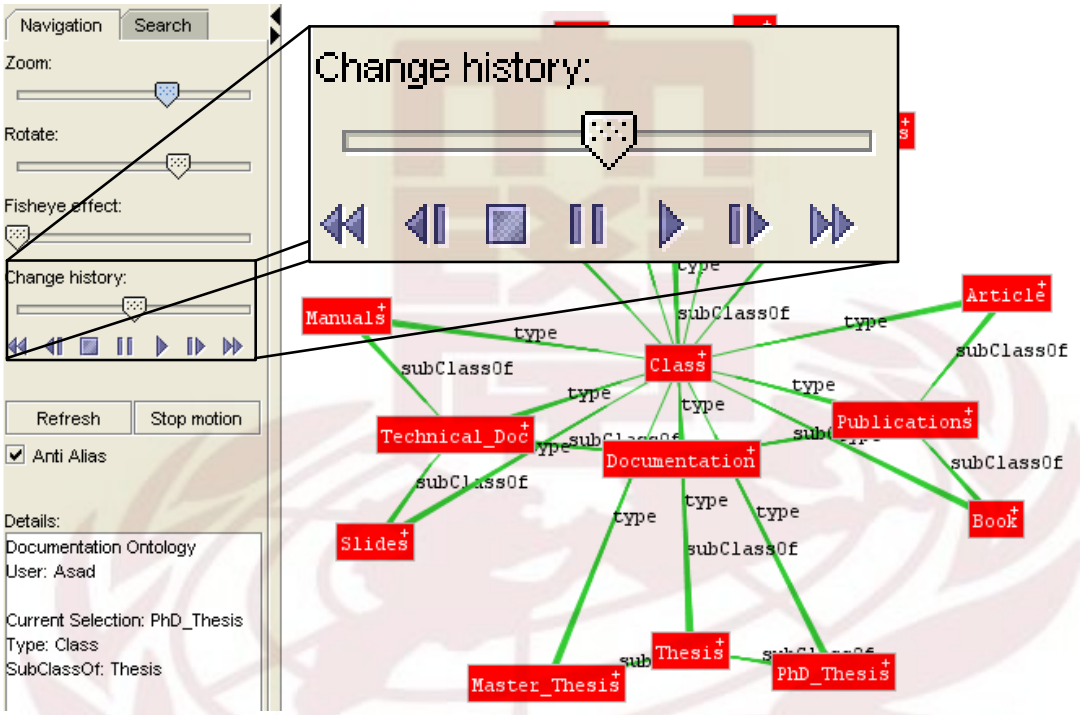


Figure 4.1: Graph visualization of ontology with change history playback feature. Users can visually navigate through the ontology changes.

4.3 Mapping Reconciliation

Mappings among systems, web services, and ontologies are required to translate query and/or share information [17, 46, 62, 67]. When ontology evolves from one consistent state to another then its mapping 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 ontologies evolved then there is also a need for re-establishment of these stalled mappings (see Figure 4.2).

Providing reliable mappings among evolving ontologies is a challenging task [89]. There is no such solution for re-engineering the broken mappings among the evolved ontologies except to completely re-establish the mappings. To re-establish the mappings among small ontologies is not an issue; however, if ontologies like Google Classification , Wiki Classification , Association for Computing Machinery (ACM) Classification Hierarchy , and Mathematics Subject Classification (MSC) Classification Hierarchy evolve with even minor changes, then complete re-establishment of mappings among such ontologies is a time consuming process. To solve this problem in time efficient manner, the introduction of *CHL* [52] containing all the changes (reason for ontology evolution), can play an important role.

Consider two ontologies are mapped and they exchange information based on the established mappings. Now one or both the ontologies are changed (evolved) to another state. In this case the already existing mappings are not reliable and also became stale. The mappings between these two ontologies also need to evolve with the evolving on-

tologies to be up to date. The scenario is discussed in two cases; (1) One of the mapped ontologies evolves, (2) Both the ontologies evolve from one consistent state to another. In both cases the mappings also needs to evolve to accommodate the mappings for the changed resources and eliminate the staleness from the already established mappings and facilitate information exchange among web services.

To reconcile the mappings in time efficient manner and remove the stalled mappings, use of CHL entries for both the ontologies have been proposed. It helps to identify the changed resources from both ontologies and establish mappings for these changed resources and update the old mapping. The need is to only extend the method for calculating Semantic Affinity (SA) by incorporating the change information from *CHL*. The details on mapping reconciliation is discussed in next chapter as well as please refer to [57].

Though this proposed process for reconciliation of mapping will reduce time; however, this raise another problem for accuracy of reestablished mappings. A need is to come up with such a technique that not only reduce the time for mapping reconciliation but also produce the same amount of accurate mappings that systems like; Falcon [46], Lily [104], AgreementMaker [19], and TaxoMap [41] generates.

4.4 Query Reformulation

Query written over one schema does not give correct results when executed over another schema [48]. It needs to be reformulated in order to fulfill the schema requirements.

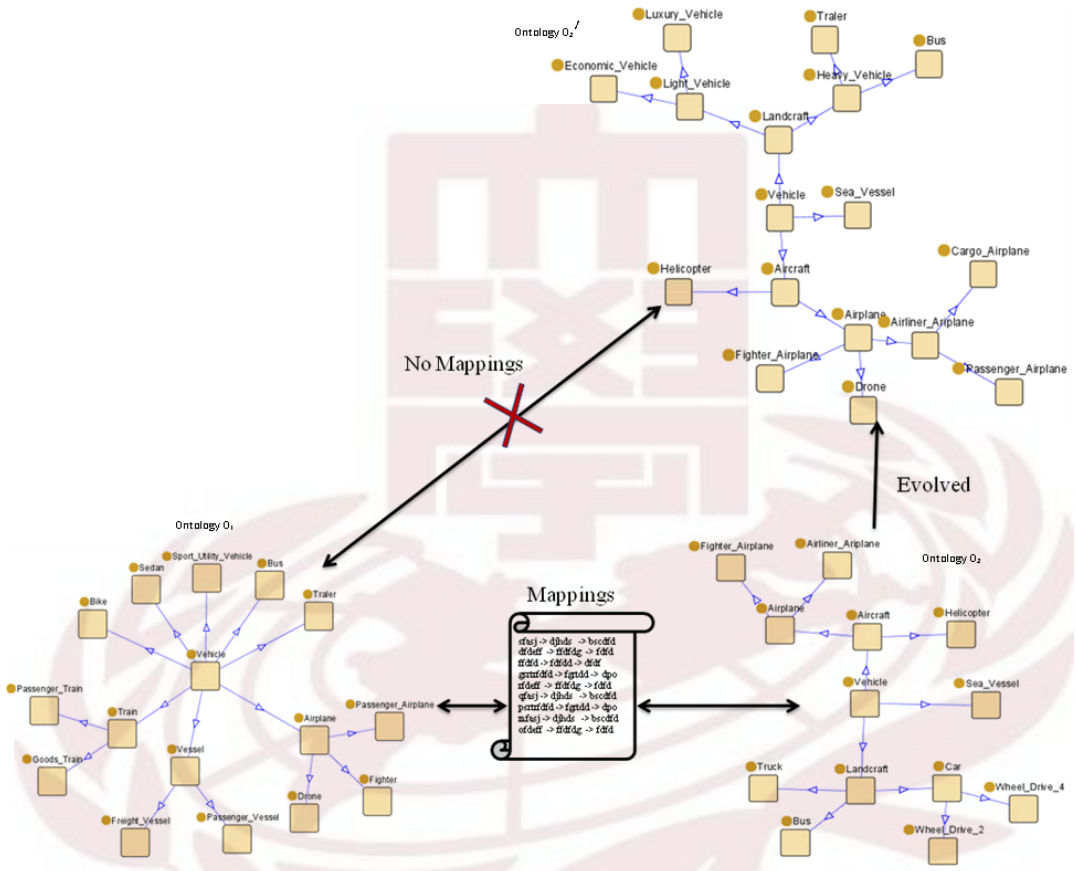


Figure 4.2: Ontology O_1 and O_2 having mappings, Ontology O_1 have evolved from state O_1 to state O_1', so the previous mappings are no more reliable as there are different changes introduced in O_1'.

Same is true for ontology, when ontology evolves from one consistent state to another then the query written over previous state needs to be reformulated to extract required results from the ontology [62]. The author in [62] 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 [62]). They evaluated the system using two different versions of Conceptual Reference Model (CRM) [21] ontology. The main idea behind this work is to maintain the changes that are reason for evolution in a repository and later used these changes for query reformulation. Very recently SPARQL Protocol and RDF Query Language (SPARQL) based push technique is proposed to broadcast notifications to change listeners [92].

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 its hard to extract the changes from the log that correspond to a particular state of ontology. *CHO* logs all the ontology changes in atomic manner and also keep the changes separate from those that correspond to different state of ontology. The notion of $ChangeSet$ has been introduced that bundles all the ontology changes together that are the cause of evolution from one state to another. So this separate $ChangeSet$ instances helps in proper reformulation of query for required version/state of ontology. Figure 3.13 of Section 3.3 is an example of $ChangeSet$ instance with the corresponding changes that cause the ontology O_1 evolved to O_1' state as shown in Figure 4.2.

Consider a SPARQL query (given in Figure 4.3) written over Ontology O_2 shown in Figure 4.2 using a web service. This query will retrieve all the instance of class Car form Ontology O_2 in descending order of their names.

```
SELECT ?technicalDocument ?documentName WHERE {
?technicalDocument a :Technical_Document .
?technicalDocument :hasName ?documentName }
ORDER BY DESC(?documentName)
```

Figure 4.3: SPARQL query for extracting TechnicalDocument instances from ontology.

Let's consider that Ontology O_2 has evolved to another state Ontology O_2', now the same query from same web service will not be able to extract the information from Ontology O_2'. For this purpose, the need is 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 in Figure 4.4. This query extracts the $ChangeSet$ instance where the changes are stored. The query given in Figure 4.5 will extract changes using the $ChangeSet$ instance information from *CHL*.

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

Figure 4.4: SPARQL query for extracting $ChangeSet$ instance.

After extracting the information from *CHL* using above queries, the first query is rewritten as given in Figure 4.6. This changed query will now get the required informa-

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

Figure 4.5: SPARQL query for extracting the changes of a $ChangeSet$ instance.

tion from the evolved state of ontology O_2'.

```
SELECT ?technicalDocument ?documentName WHERE {
?technicalDocument a :TechnicalDocument .
?technicalDocument :hasName ?documentName }
ORDER BY DESC(?documentName)
```

Figure 4.6: SPARQL query for extracting the changes of a $ChangeSet$ instance.

4.5 Change Prediction

For conflict resolution as well as for future change prediction, the logged changes can be of good help. The changes logged in *CHL* [52] are simple changes that do provide lots of open space for change patterns. See Figure 4.7, after every new class addition a class renaming changes are due. The same way after every property addition, property renaming and setting its domain and range is occurring. So these patterns can be extracted and can be used in conflict resolution and next change prediction that indirectly help the dependent applications and services. Using these frequent patterns can help in having a better understanding of the ontology development. Figure 4.7 is an example of mining frequent patterns from some of the changes logged.

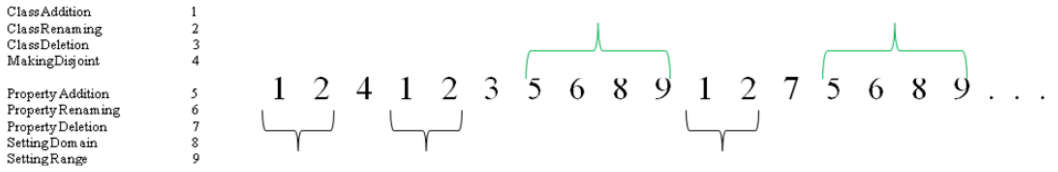


Figure 4.7: Mining frequent change pattern from logged changes.

4.6 Collaborative Ontology Engineering

The ontology as by definition are shared, so changes made to any instance of ontology should also be reflected to all other instance of an ontology. To support this 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 4.8 where the changes of O_w and O_x are propagated to O_1, O_2, and O_3.

In this chapter, potential applications of *CHL* are discussed. In addition, the suggested solutions are also presented that can be used to minimize the effects of ontology evolution on the dependent data, applications, systems, and services. Next chapter, mainly focus on the procedure of capturing ontology changes during ontology evolution (i.e., during a change session). In addition, the procedure for mapping reconciliation based on the captured changes is also presented in detail.

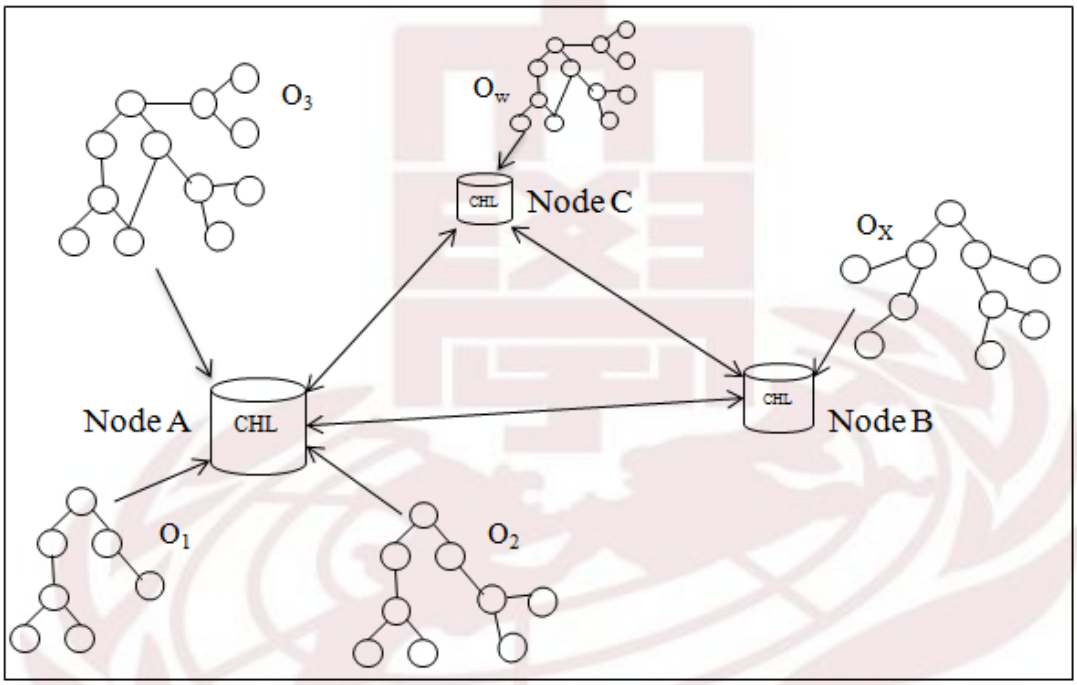


Figure 4.8: This figure represent the scenario of collaborative ontology engineering and the use of CHL in this scenario.

Chapter 5

Change Capturing and Mapping Reconciliation

This chapter explains the process of change listening and logging in *CHL*. Ontology change capturing process is responsible for capturing all the ontology changes and logging them in *CHL*. The captured changes are then used for reconciling the stalled mapping in a performance efficient manner. The detail procedure for mapping reconciliation is presented in this chapter. For example (see Figure 3.13), where different changes including $ClassAddition$ and $RangeAddition$ instances are logged in *CHL*. These logged changes are then used for the purpose of mapping reconciliation. The final outcome is the reconciled mapping with better performance than the existing systems. Although, the reference implementation of *CHL* is coupled with mapping reconciliation; however, it is designed to be independent of change log implementations.

5.1 Change Capturing

As discussed in Chapter 2, change traceability and mapping reconciliation are essential ingredients of ontology change management systems. A semantic structure for representing ontology changes is presented in Chapter 3 and a comprehensive workflow for change traceability and mapping reconciliation is presented in this chapter. Consequently, this chapter is organized into two major sections: In the first section, Change Capturing procedure is introduced that detect ontology changes of a change session and log it in *CHL*. The second section of this chapter explains the detail mapping reconciliation procedure for design and development. For better understandability, the algorithmic workflow is also provided.

The proposed framework has been envisioned as an enabling component for ontology editors. The framework itself does not provide ontology editing services, rather it implements listeners (specific to an ontology editor) to monitor and log changes. The framework is implemented for ontologies defined in *Resource Description Framework Schema (rdfs)* and all variants of *Web Ontology Language (OWL)*. Various components are implemented in the framework to perform tasks related to change history management. For example, the Change Logger component preserve the changes. The component based framework architecture is given in Figure 5.1, whereas a detailed description of these components is given in [51]. To validate the working of the proposed framework, a *Tab-Widget* plug-in, $ChangeTracer$ Tab, for ontology editor i.e., Protege has been developed. Details of various modules are given below.

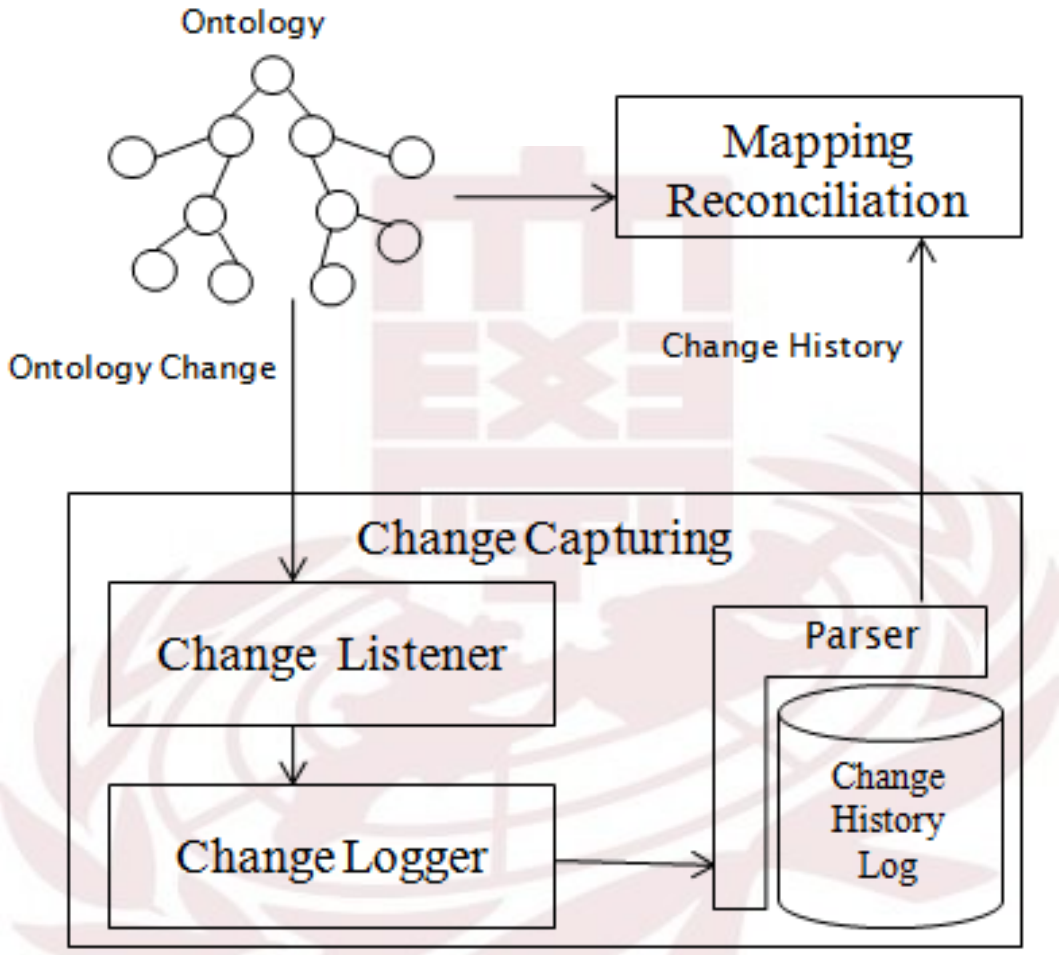


Figure 5.1: The figure shows overall architecture of the proposed system for change capturing and reusing the changes for mapping reconciliation.

5.1.1 Change Listener

The Change listener module consists of multiple listeners which actively monitor various types of changes applied to the ontology model in Protege. Table 5.1 presents all the listeners implemented using the Protege OWL API. *ProjectListener* listens for project related changes. One of its main functions is to listen for save or close commands to save the active $ChangeSet$ instance in *CHL*. *KnolwedgeBaseListener* is the most used listener for capturing the changes which are also triggered by the other listeners. This listener overlaps with *ClsListener, ClassListener, SlotListener, and PropertyListener*. *FacetListener* also overlaps with the *ClsListener, ClassListener, SlotListener, and PropertyListener*; however, it also provides additional axiom related change information. The *InstanceListener* overlaps with the *KnolwedgeBaseListener* for capturing the instance level changes. When a change is committed, its corresponding listener collects the necessary contextual information, such as change agent, target, and updated value.

5.1.2 Change Logger

The changes captured by the listeners are logged with conformance to *CHO*. All the changes are handled at the atomic level. This aspect covers both atomic changes, such as deleting a single concept, as well as complex scenarios e.g., deleting a sub-tree involving multiple concepts. Atomic change are easy to handle. In contrast, compound changes are sometimes harder to implement. For instance, deleting a $ChangeAgent$ class will also impact its subclasses. As a result, every change requests has to be handled at atomic level. Logging component ultimately plays a critical role in maintaining atomicity of changes, and undo or redo operations, in case of a failure. Logging component uses *CHO* specifi-

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

Table 5.1: List of change listeners implemented in the Change Capturing plug-in to listen and log ontology changes.

cations for persistent storage of changes.

5.1.3 Parser

The job of Parser is to: 1) Parse *CHL* for all the $ChangeSet(s)$ instances that correspond(s) to the open model in Protege on user request. 2) Parser is also used to communicate with the *CHL* and stores the changes. 3) The parser is also responsible for extracting the changes from *CHL* for the mapping reconciliation procedure.

Figure 5.2 shows the SPARQL queries for parsing the ontology changes form *CHL*. These queries are executed on the *CHL* from *Parser* module to extract all the changes corresponding to a specific $ChangeSet$ instance. The first query extracts the $ChangeSet$ instances in time order, and then based on user needs, appropriate $ChangeSet$ instance is selected and its corresponding changes are extracted from the *CHL*. Afterwards the details of these changes are extracted. The queries are important for the purpose to extract the changes from *CHL*, which are used in mapping reconciliation.

5.2 Reconciliation of Ontology Mappings

Mappings are defined between two ontologies at a time; one is called the source ontology and the other being called the target ontology. The proposed scheme for mapping reconciliation in dynamic/evolving ontologies is time efficient and eliminates staleness from the mappings. It uses ontology changes of evolving ontologies to reconcile the mappings. It is based on the concept of *CHL* [52] that contains all the ontology changes that hap-

pen to an ontology during evolution. The change log is required for the reason to know which of the ontology resource has changed and resulted in unreliable existing elements in mappings. The unreliable mappings are because of the changed resources and need re-alignments. For this reason, the changes in dynamic ontology need to be maintained persistently for their later use, such as query reformulation, ontology recovery, and change prediction [54], and in this case for mapping reconciliation. So basically the proposed scheme (for architecture see Figure 5.3) for reconciliation of ontology mapping has two main components; 1) Change History Log to maintain all the ontology changes in a semantic structure and 2) Mapping Reconciliation process to eliminate unreliable mappings from the existing mapping and re-establish mapping for the dynamic ontologies.

5.2.1 Mapping Reconciliation Procedure

As discussed above, there are different algorithms available to establish mappings between ontologies [5, 17, 46, 67, 76, 104]. The existing systems do provide the facility for re-establishment of mappings between dynamic ontologies after their evolution; however, they start the mapping re-establishment process from the scratch which is time consuming operation. The proposed contribution is to use the change entries of ontology (after evolution) stored in the *CHL* [52] for reconciliation of mappings between ontologies, which not only helps to eliminate staled mappings but also takes less time to reconcile mappings in dynamic/evolving ontologies. In this approach, the focus is to only concentrate on the changed resources between the evolved ontologies. The approach is most suitable for large sized ontologies having hundreds and thousands of resources, for example; when reconciling mappings (after change) among *Mrinkman, GTT, GEMET, NALT,*

(**Algorithm 1 - Mapping Reconciliation Algorithm**) Mapping reconciliation algorithm for ontology mapping using ontology changes stored in *Change History Log (CHL)*.
Input: Ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ for mapping reconciliation.
Input: Ontology change information (i.e., Δ_1 and Δ_2) from *CHL* of both ontologies, i.e., $\Delta_1 \in \mathcal{O}_1$ and $\Delta_2 \in \mathcal{O}_2$.
Constant: A resource matching threshold is defined as ψ = 0.70
Output: Set of mappings for the changed resources and then updated in the original mappings file.
1. /* Check for change of resources in CHL of both the mapped ontologies and read the changes in Δ*/
2. **if** $\exists\Delta \sqcap \exists\Delta.\mathcal{O}_1.CHL.NewChanges$ **then**
3. /* Read the changes in Δ_1 */
4. $\Delta_1 \leftarrow \{x|\langle CHL_\Delta, x\rangle Change\}$
5. **endif**
6. **if** $\exists\Delta \sqcap \exists\Delta.\mathcal{O}_2.CHL.NewChanges$ **then**
7. /* Read the changes in Δ_2 */
8. $\Delta_2 \leftarrow \{x|\langle CHL_\Delta, x\rangle Change\}$
9. **endif**
10. /* Delete all the mappings from the original mapping file that are subject to change because of change in the mapped resources. This method takes both Δ_1 and $Delta_2$ as optional parameters and use if a change exist in CHL and retrieved in Δ */
11. $Execute.delete(Mappings, [\Delta_1], [\Delta_2])$
12. /* Start mapping reconciliation procedure by calculating semantic affinity */
13. **if** $\exists\Delta_1.Change \sqcap \exists\Delta_2.Change$ **then**
14. /* Calculate semantic affinity using changed resources of both the ontologies */
15. $R - Map[][] \leftarrow SemanticAffinity(C_1 \in \mathcal{O}_1, \Delta_1, C_2 \in \mathcal{O}_2, \Delta_2, \psi)$
16. **else-if** $\exists\Delta_1.Change \sqcup \exists\Delta_2.Change$ **then**
17. /* Calculate semantic affinity using changed resources of one changed ontology represented as Δ' */
18. $R - Map[][] \leftarrow SemanticAffinity(C_1 \in \mathcal{O}_1, C_2 \in \mathcal{O}_2, \Delta', \psi)$
19. **endif**
20. /* Update the original mapping file with the reconciled mappings for the changed resources.*/
21. $Execute.delete(Mappings, R - Map[][])$
22. **end**

Google Classification, Wiki Classification, ACM Classification Hierarchy, and MSC Classification Hierarchy. The larger the size of ontology the better and more time efficient the approach is against any of the above discussed algorithms. Detail procedure is given below.

5.2.2 Re-establishing Mappings

Consider the scenario given in Figure 5.3, where two ontologies are mapped and they exchange information based on the established mappings. Now suppose that one or both ontologies change (evolve) to another state (see Figure 5.3). In this case the already existing mappings are of no more use as they are not reliable and also became stale. So the mappings between these two ontologies need to evolve with the evolving ontologies to be up to date. To elaborate it further, two different cases are considered.

Case 1: If one of the ontologies evolves from one state to another then its mapping with other ontologies becomes unreliable as there will be a definite change in the resources mapped with the other ontology. To reconcile the mappings between these ontologies, mappings should be reconciled. Instead of completely re-initializing the mapping process from the scratch, which is a time consuming process. *CHL* entries are used to figure out the changed resources in the evolved ontology. Then use only these changed resources from *CHL* in mapping reconciliation process to map it with the other ontology and simply update the previous mappings with the new one and remove the staled mapping entries. In this case, the need is to only extend the method for calculating the Semantic Affinity (SA) by incorporating the

$SA(C_1, C_2, \Delta_2, \psi)$ C_1 Resource from Ontology $\mathcal{O}_1$
C_2 Resource from changed Ontology $\mathcal{O}_2$
Δ_2 Change information from CHL of Ontology $\mathcal{O}_2$
ψ User defined threshold for resource match

change information from *CHL*. So the modified method, including parameters, is given below;

Case 2: Consider the second case where both the ontologies evolve from one consistent state to another as demonstrated in Figure 5.3. This is also the worst case scenario in terms of execution time for mapping reconciliation. In this case the mapping also needs to evolve to accommodate the mappings for the new changed resources and eliminate the staleness from the already established mappings. Again, there is no need to completely re-establish the mappings between both ontologies like existing systems which is a time and resource consuming process; instead, the need is to reconcile mappings for the changed resources. As given in Figure 5.3, both ontologies O_1 and O_2 have evolved. To reconcile the mappings between the evolved ontologies and remove the staled mappings in time efficient manner, the *CHL* entries are used for both ontologies to identify the changed resources from them. Based on identified changes, mappings are reconciled for these changed resources, the old mappings are updated, and the unreliable (staled) mappings from the previous mappings are removed. This is not only a time efficient technique but also eliminates the staleness from the mappings that need to be updated for reliable communication and exchange of information between systems and/or services.

The inputs for this module are; (also shown in Figure 5.3) the evolved ontologies

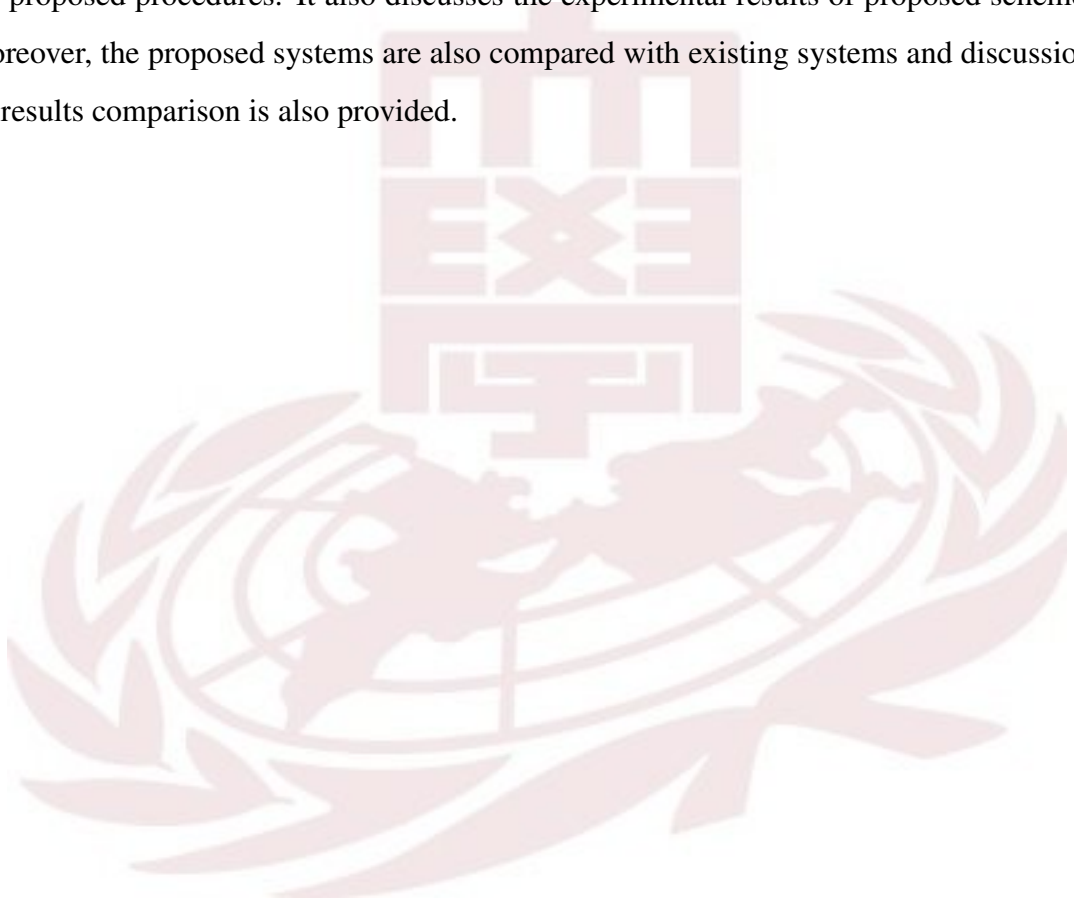
$SA(C_1, \Delta_1, C_2, \Delta_2, \psi)$ C_1 Resource from Ontology $\mathcal{O}_1$
Δ_1 Change information from CHL of Ontology $\mathcal{O}_1$
C_2 Resource from Ontology $\mathcal{O}_2$
Δ_2 Change information from CHL of Ontology $\mathcal{O}_2$
ψ User defined threshold for resource match

O_1 and O_2, the *CHL* entries for both ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ for ontology O_1 and for ontology O_2 respectively. The pervious mappings between these two ontologies are also updated in the proposed algorithm (see Algorithm 1) execution. The SA is calculated by incorporating the change information from the *CHL*. So the modified method including parameters is like given below.

Δ_1 and Δ_2 are the changes of both ontology contained in the *CHL*. For calculating the SA, these changes are required and extracted from the *CHL* using SPARQL queries given in Figure 5.2. To get the latest changes, first the $ChangeSet$ instances are extracted and sorted in descending order of their $timestamp$ defined in the *CHO* and the top most $ChangeSet$ instance is selected. Afterwards, all the changes corresponding to the selected $ChangeSet$ instance are retrieved from the *CHL*.

After the process of reconciliation, the staled parts of mappings are removed from the overall mappings. It is updated with the new changed mappings, as shown in Figure 5.3, with color blue in the mappings and underlined. This process of reconciliation of mappings not only eliminates the staleness from the mappings but is also more time and memory efficient (as it just focuses on the changed resource) that makes it more suitable for systems and services dealing in information exchange.

This chapter focused on the design and development of a change capturing procedure. The captured changes are logged in *CHL*. Later the changes are retrieved from *CHL* and used for mapping reconciliation. The procedure for mapping reconciliation is also presented in detail with two scenarios. The next chapter is on the implementation of the proposed procedures. It also discusses the experimental results of proposed scheme. Moreover, the proposed systems are also compared with existing systems and discussion on results comparison is also provided.



```
SELECT  ?changes  ?timeStamp
WHERE { ?changes docLog:isPartOf changeSetInstance .
?changes docLog:hasTimeStamp ?timeStamp }
ORDER BY DESC(?timeStamp)

SELECT  ?changedTarget ?isSubClassOf
WHERE { Resource docLog:hasChangedTarget ?changedTarget .
Resource docLog:isSubClassOf  ?isSubClassOf }

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

Figure 5.2: SPARQL query for extracting changes corresponding to the $ChangeSet$ and then extracting their relevant details.

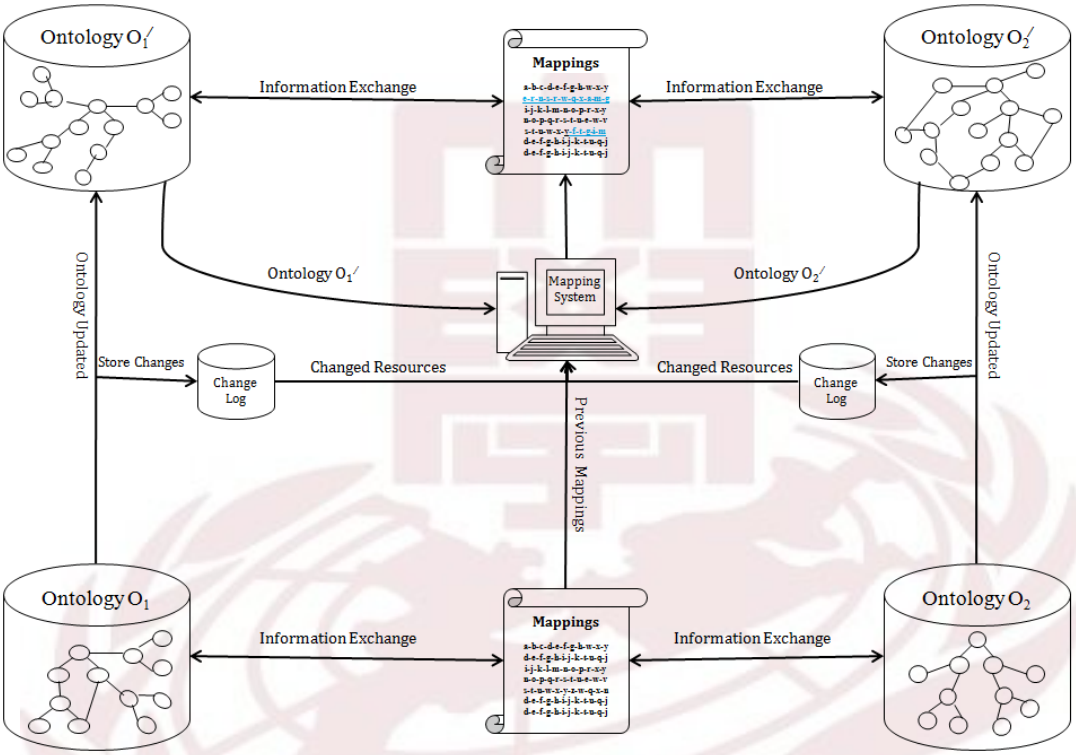


Figure 5.3: Shows the overall framework for reconciliation of mappings in dynamic/evolving web ontologies. It also shows the process of ontology change storage and reuse for mapping reconciliation process.

Chapter 6

Implementation and Results

This chapter provides the details on building a framework for tracing ontology changes. This helps in automatically detecting and logging all the changes, triggered by the change request from ontology engineer. The focus is to develop a generic framework for temporal traceability of ontology changes during evolution. For this a repository is required to record all the changes. For this a semantic structure for representing ontology changes is developed to keep track of changes. It can also provide facility to make the ontology changes temporally traceable. The repository helps in providing mapping reconciliation facility discussed in Section 6.2.

The proposed framework is implemented as a plug-in for ontology editing tool protege [78]. There are different ontology tools available, as discussed in Chapter 2, Table 2.2 and also discussed in [13, 27]. The selection of plug-in development for protege is based on the analysis of the existing tools and the facilities available in them. Keeping the observations in mind, the opinion is on Protege's simplicity of use, flexibility, richness, effectiveness, and its adoptability by research community [13, 27, 78]; it is the best suited

tool to be considered.

6.1 Change Capturing

The *Bibliography* ontology is used for the development and testing of the plug-in. First of all, the results of the plug-in for change capturing are provided using the *Bibliography* ontology.

There exists systems that also facilitate the change capturing facility to compare the developed plug-in with. However, these systems have some limitations for capturing all the changed information. The existing systems, such as a Protege plug-in $ChangesTab$ [66], another Protege plug-in $VersionLogGenerator$ [84], $ChangeDetection$ [84], and $ChangeCapturing$ [81] provide the change capturing facility. The proposed plug-in ($ChangeTracer$) is compared with $ChangesTab$, $VersionLogGenerator$, $ChangeDetection$, and $ChangeCapturing$ to analyze its change capturing capability. For this, 35 different changes covering all four different categories (i.e., Change in Hierarchy, Change in Class, Change in Property, and Other Changes) were made to the *Bibliography ontology*. $ChangeTracer$, $ChangesTab$, and $VersionLogGenerator$ were configured with Protege, $ChangeCapturing$ with NeOn Toolkit (http://www.neon-toolkit.org/), and $ChangeDetection$ was used as a stand alone application. Out of these 35 changes, $ChangesTab$ captured 28 changes, $VersionLogGenerator$ captured 28 changes, $ChangeDetection$ captured 33 changes, $ChangeCapturing$ captured 32 changes, whereas the proposed plug-in i.e., $ChangeTracer$ captured 32 changes. The graph representing these results is given in Fig. 6.1, where the y-axis represents the number of changes captured and the

x-axis represents the total number of changes made.

Protege internally implements different listeners (see Table 5.1) that report when a change occurs in the open ontology model. ($ChangeTracer$), $ChangesTab$, and $VersionLogGenerator$ implement these listeners and capture the changes that are triggered by Protege. However, when certain events are triggered, such as element deletion (i.e., Class, Property, Individual), then the element is first deleted and later the event is notified. Due to this reason, the deleted element's information is missed and not captured by the plug-ins ($ChangeTab$ and $VersionLogGenerator$). This also happens with $ChangeCapturing$.

In the proposed plug-in, to handle this issue, difference is computed for the new model and the old model at the time of element deletion event, which provides the information about the deleted element. Another issue with Protege is that for Datatype property range addition, deletion, and modification, there is no event notification. So these changes are misidentified. As visible from the Figure 6.1, that $ChangeDetection$ captured more changes than the others; however, its performance heavily depends on the types of changes. For example, if the changes are always of element modification in an ontology then the $ChangeDetection$ will misidentify them. Moreover, $ChangeDetection$ cannot identify a sequence of changes.

To get more concrete results, the experiment are repeated for 20 times with 35 different and random changes. Details and results of these experiments are given in Table 6.1 and its graph representation is given in Fig. 6.2. Total of 700 different changes were made. Out of these 700 changes, 663 (i.e., 94.71%) changes were detected by

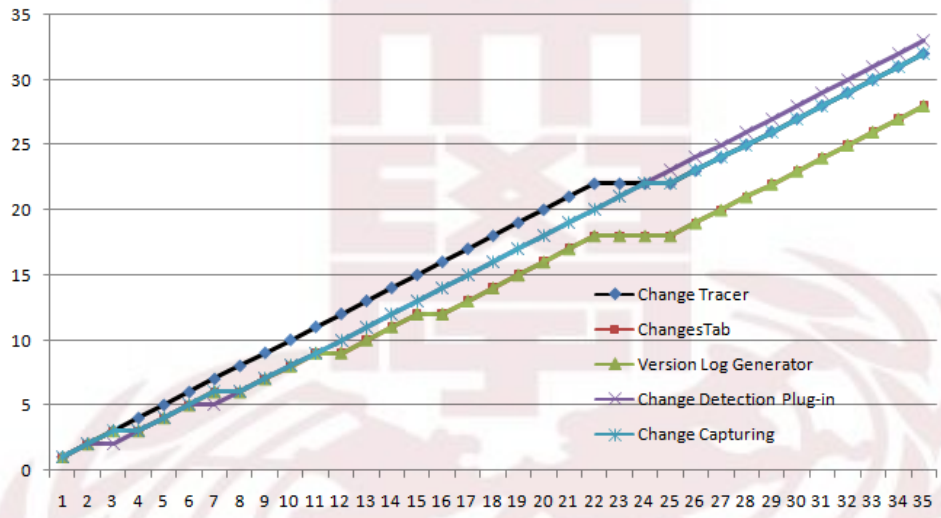


Figure 6.1: Comparison of $ChangeTracer$ against $ChangesTab$, $VersionLogGenerator$, $ChangeDetection$, and $ChangeCapturing$. X-axis represents the number of changes applied to ontology, whereas, y-axis represents the changes captured by the respective systems.

$ChangeTracer$, 632 (i.e., 90.28%) changes by $ChangesTab$, 645 (i.e., 92.14%) changes by $VersionLogGenerator$, 629 (i.e., 89.86%) changes by $ChangesDetection$, and 667 (i.e., 95.28%) changes by $ChangesCapturing$. The results clearly show that the proposed plug-in (i.e., $ChangeTracer$) has outperformed the $ChangesTab$, $VersionLogGenerator$, and $ChangeDetection$ in terms of change capturing. In comparison with $ChangeCapturing$, the proposed system has almost the same results. However, against the $ChangeCapturing$, the proposed plug-in uses *difference()* method of *Model* class from *Jena API* to capture the missing changes, whereas, the $ChangeCapturing$ always misses certain changes [81].

It is also very important to capture all the changes in an ontology not only to make the information model complete, in addition, if required to use the changes for recovery or mapping reconciliation purpose then the complete set of changes applied are necessary. For this purpose, an exhaustive testing of the proposed plug-in on Ontology Meta Data Vocabulary (OMV) ontology [43], Semantic Web Research Community (SWRC) ontology [96], Conceptual Reference Model (CRM) ontology [21] and the standard dataset of Semantic Web Technology Evaluation Ontology (SWETO) [70] is provided in Appnedix A, Section A.1.

6.2 Mapping Reconciliation

This section discusses in detail about the results achieved for mapping reconciliation based on the changes captured and logged in *CHL*. Using *CHL*, the existing mapping systems are extended to use the change entries for mapping reconciliation. The extensions are proposed to the existing mapping systems, i.e., Falcon [46], H-Match [17],

Specifications	Change Tracer	Changes Tab	Version Log Generator	Change Detection Plug-in	Change Capturing
No of Experiments	20	20	20	20	20
No of Changes per Experiments	35	35	35	35	35
Total Changes	20 * 35 = 700	20 * 35 = 700	20 * 35 = 700	20 * 35 = 700	20 * 35 = 700
Changes Captured	663	632	645	629	667
Average	94.71	90.28	92.14	89.86	95.28

Table 6.1: Comparative Analysis of Change Detection Approaches i.e., $ChangesTab$, $VersionLogGenerator$, $ChangeDetectionPlug-in$, and $ChangeCapturing$ against the proposed $ChangeTracer$.

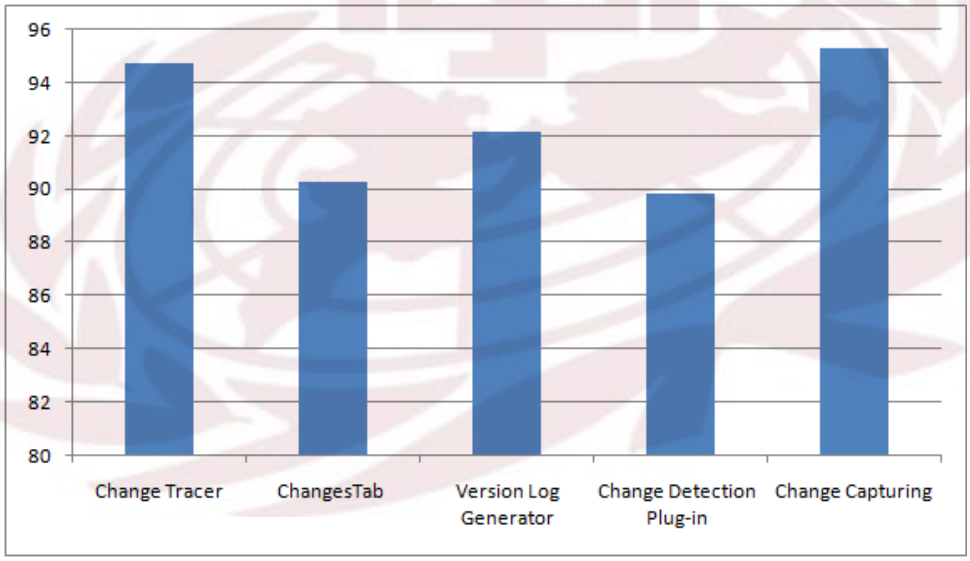


Figure 6.2: Shows the average result of 20 experiments with 35 different changes using $ChangeTracer$, $ChangesTab$, $VersionLogGenerator$, $ChangeDetection$, and $ChangeCapturing$.

FOAM [22], Lily [104], AgreementMaker [19, 20], Prompt [76], and TaxoMap [41]. The results comparison of the proposed extensions against the existing systems verifies that the amount of time required for reconciliation of mappings using the proposed extensions is far less than the existing systems. The data sets used in these experiments are; *Mouse, Human, Mrinkman, GTT, GEMET, and NALT ontologies*, available online at (http://oaei.ontologymatching.org/). Other data sets such as *Health* and *Food* ontology , *People+Pets* ontology , *ACM* and *Springer* ontology, *HL7 Classes* ontology, and *openEHR Classes* ontology have also been used for systems' detail comparison.

The experiments are all conducted on a machine with 2.66 GH Quad Core processor and 4 GB of main memory. The experiments are carried out for both cases explained in Section 5.2 (i.e., Reconciliation of Ontology Mappings). In all the experiments, a constant similarity value of ψ 0.70 is kept as a matching threshold. Numbers of iterations in most of the systems are kept as default but in case of FOAM [22] it is set to 7 iterations per execution; however, it does not affect the results as systems are not compared with one another for accuracy. These experiments are by no means the comparison of existing systems, but are in fact the comparison of each individual system with the proposed extensions to that individual system.

The experiments are conducted in two modes, i.e., changes are considered at complex and at atomic level [54]. Complex change is a change that consists of several atomic level changes e.g., deletion of super class will result in complex change including the deletion of all the subclasses of that super class. Atomic change is a simple change e.g., renaming a resource. In these experiments, changes are mostly the introduction of new resources in the domain ontologies. Figure 6.3 is the confirmation of issue identified in this research

and it uncovers the limitations of the existing systems by not focusing on mapping evolution and its effects. The existing systems take more time for regenerations of mappings with both; complex changes and atomic changes. 25 complex changes in each version of ontology used in the experiments shown in Figure 6.3-A are introduced, while the atomic changes for each ontology version used in Figure 6.3-B are given in Table 6.4.

6.2.1 Comparison using Complex Changes

To test the existing systems with the proposed extensions to the systems, 25 random changes (complex) are introduced to different ontologies used for the experiments. These changes made the ontologies (listed in Table 6.2) evolve from one consistent state to another. In these experiments, the ontologies are considered in full i.e., the structure and with its instances. As discussed above, 25 complex changes are made to every version of the ontologies effecting both structure and instances. The existing algorithms (mentioned above) and proposed extensions to these algorithms are all tested for both cases/scenarios.

Case 1: In this scenario, only one of the ontology evolved from one state to another while the second ontology remained unchanged. Falcon, H-Match, Lily, and TaxoMap are first checked for initial mapping between the ontologies, then for re-establishment of the mappings due to the changes in ontology. Afterwards, the proposed extensions were applied for the changed ontologies to reconcile the mappings. As discussed earlier, the existing algorithms start from scratch, so these systems mostly they take more time than the previous mapping process as shown in Table 6.2. The proposed extension to existing algorithms using *CHL* [52], only considers the

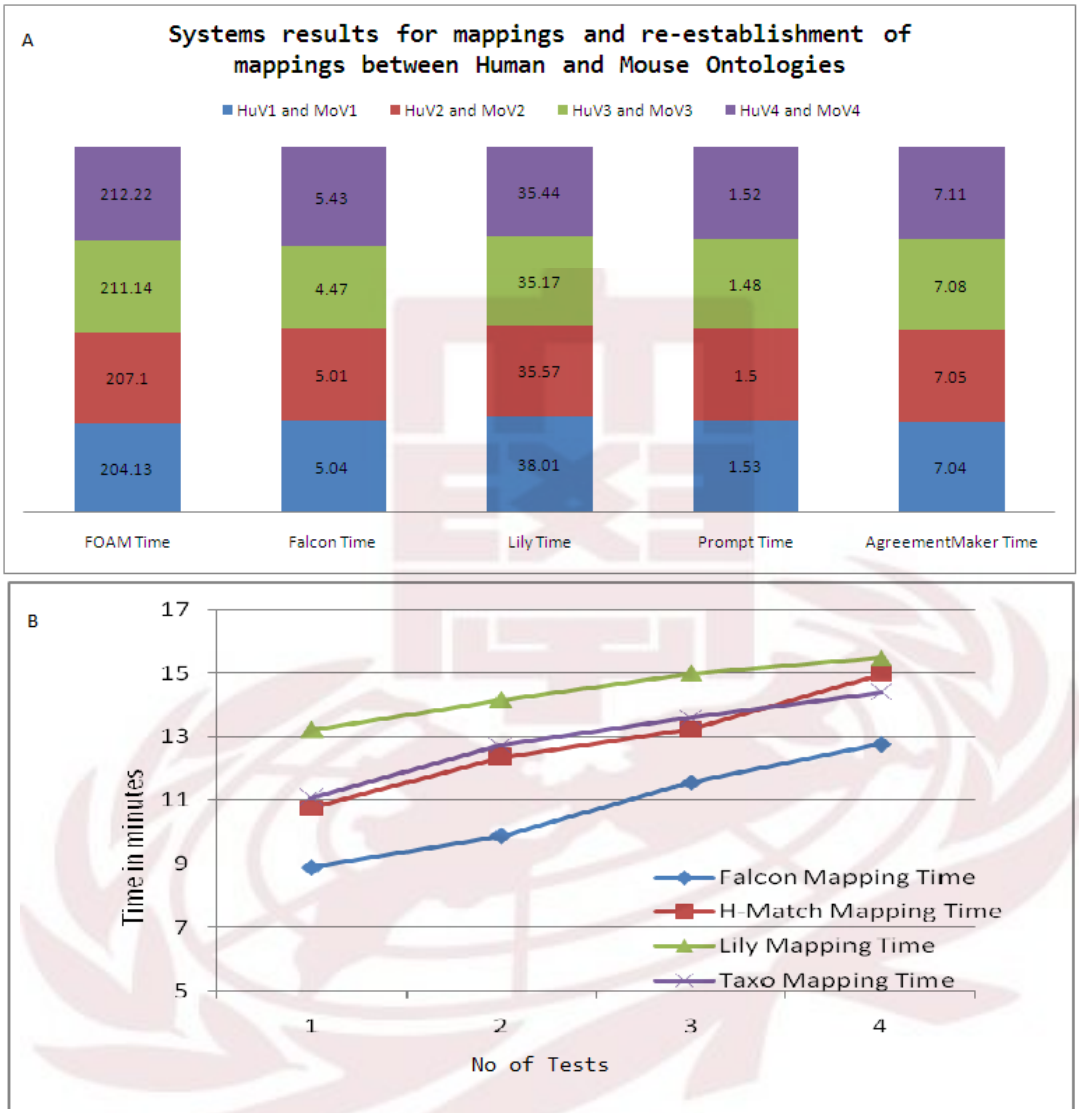


Figure 6.3: Figure 6.3-A shows the mapping and Re-establishment of mapping results with respect to time for Mouse and Human ontology using Falcon [46] and H-Match [17] with complex changes. Figure 6.3-B shows the Mapping and Re-establishment of mapping results with respect to time for Mouse and Human ontology using FOAM [22], Falcon [46], Lily [104], AgreementMaker [19], and Prompt [76] with atomic changes.

changed resources and reconciles mappings for the changed resources. Proposed extensions (see Table 6.2) showed better performance than the existing systems. The amount of the computation time (shown in Table 6.2). The last set of four rows in Table 6.2 shows better performance of proposed extensions against the existing systems. The results show that extensions using the *CHL* drastically reduced the computation time for reconciliation of mappings in dynamic ontologies.

Case 2: In this scenario, both ontologies evolved from one consistent state to another. Falcon, H-Match, Lily, and TaxoMap are first checked for initial mappings between the ontologies, then for re-establishment of mappings due to the changes in both ontologies. The algorithms are tested again for re-establishment of mappings and then their implementation with the proposed extensions. These algorithms start from scratch so they take more time than the previous test (shown in Table 6.3) as against the proposed extensions that only consider the changed resources and reconcile mappings for the changed resources. Proposed technique helped in saving large amount of the computation time (shown in Table 6.3). The set of 4 last rows in Table 6.3 gives the evidence of better performance against the existing systems. The results in both Table 6.2 and Table 6.3 show that extensions using the *CHL* reduce the computation time for reconciliation of mappings for both cases in dynamic web ontologies.

Mapping Systems	**Mouse Onto vs. Human Onto**	**Brinkman Onto vs. GTT Onto**	**GEMET Onto vs. NALT Onto**
Falcon Mapping Time	8.89 m	32.40 m	51.33 m
H-Match Mapping Time	10.76 m	39.13 m	1.12 h
Lily Mapping Time	13.22 m	34.03 m	52.65 m
TaxoMap Mapping Time	11.08 m	33.41 m	52.65 m
Falcon Re-Mapping Time	9.17 m	33.36 m	52.43 m
H-Match Re-Mapping Time	11.86 m	40.47 m	1.14 h
Lily Re-Mapping Time	14.15 m	35.51 m	52.91 m
TaxoMap Re-Mapping Time	12.73 m	34.09 m	53.76 m
Extended Falcon Re-Mapping Time	1.08 m	3.11 m	5.36 m
Extended H-Match Re-Mapping Time	1.42 m	2.78 m	7.31 m
Extended Lily Re-Mapping Time	1.93 m	4.08 m	6.73 m
Extended TaxoMap Re-Mapping Time	1.87 m	3.64 m	5.92 m

Table 6.2: Computation time analysis of Falcon [46], H-Match [17], Lily [104], and TaxoMap [41] for mapping, re-mapping, and reconciliation of mappings with proposed extensions to Falcon, H-Match, Lily, and TaxoMap using Change History Log when one of the mapped ontologies changes.

Mapping Systems	**Mouse Onto vs. Human Onto**	**Brinkman Onto vs. GTT Onto**	**GEMET Onto vs. NALT Onto**
Falcon Mapping Time	8.89 m	32.40 m	51.33 m
H-Match Mapping Time	10.76 m	39.13 m	1.12 h
Lily Mapping Time	13.22 m	34.03 m	52.65 m
TaxoMap Mapping Time	11.08 m	33.41 m	52.65 m
Falcon Re-Mapping Time	9.87 m	34.63 m	53.71 m
H-Match Re-Mapping Time	12.35 m	41.55 m	1.17 h
Lily Re-Mapping Time	15.43 m	37.20 m	54.97 m
TaxoMap Re-Mapping Time	13.21 m	35.93 m	55.36 m
Extended Falcon Re-Mapping Time	2.36 m	5.06 m	9.48 m
Extended H-Match Re-Mapping Time	2.96 m	4.88 m	12.39 m
Extended Lily Re-Mapping Time	3.45 m	6.75 m	10.37 m
Extended TaxoMap Re-Mapping Time	2.97 m	6.09 m	10.18 m

Table 6.3: Computation time analysis of Falcon [46], H-Match [17], Lily [104], and TaxoMap [41] for mapping, re-mapping, and reconciliation of mappings with proposed extensions to Falcon, H-Match, Lily, and TaxoMap using Change History Log when both the mapped ontologies changes.

Ontology Versions	**Human**	**Mouse**	**Health**	**Food**	**People+Pet**	**ACM Ontology**	**Springer Ontology**
Version1	Original	Original	Original	Original	Original	Original	Original
Version2= Version1 + Changes	283	166	169	122	120	109	176
Version3= Version2 + Changes	112	201	153	161	172	133	114
Version4= Version3 + Changes	123	198	145	114	109	141	106

Table 6.4: Ontology versions and the number of atomic changes applied to one version that transforms ontology to another version. All the ontologies are listed in 1^{st} row. Numeric values are the number of changes of current version against the previous version of ontology.

6.2.2 Comparison using Atomic Changes

This section describes the experimental results when the data sets with changes at atomic level are tested with the existing systems and with proposed extensions to the existing systems. For these experiments, only the structures of ontologies are considered for the mapping procedures and no individuals (instances) are used. Table 6.4 shows different versions of data sets and the number of atomic level changes between these versions. The existing systems, i.e., Falcon [46], FOAM [22], Lily [104], AgreementMaker [20], Prompt [76], and the proposed extensions are tested on these data sets for the following two cases.

Case 1: In the first case, only one of the mapped ontology evolved from one state to another while the second ontology remained unchanged. The existing systems were

first checked for initial mappings between the ontologies and for re-establishment of mappings after the changes in the ontology. Afterwards, the proposed extensions are applied for mapping reconciliation between the changed ontology. The existing systems and proposed extensions are all tested in detail using the data sets given in Table 6.4 and their results for Case 1 are shown in Figure 6.4. The execution time of these systems varies against one another (see Figure 6.4) due to different matching schemes used in their implementation. Execution time shown in Figure 6.4 is all in minutes and fractions of minutes. Each graph of Figure 6.4 shows the results of existing systems and the proposed extensions on a particular data set with its different versions.

Each graph of Figure 6.4 consists of 5 pairs, making 10 bars in total. Each alternative pair is the results comparison of the proposed system against the existing system. 1^{st} bar of each pair shows the execution time of the existing system on each version (differentiated using colors) of the ontology, while the 2^{nd} bar of each pair shows the execution time for the proposed extensions for different versions of the ontology. One very obvious pattern visible in each graph of Figure 6.4 is that the execution time of proposed extensions on starting versions of ontologies is always same or a fraction greater (max by 24 seconds) than the existing systems. It is because, if the ontologies are matched for the first time, in this case, the proposed system carries out complete mapping procedure in addition to looking for the changes from the *CHL* and existing mappings. The detailed experimental results shown in Figure 6.4 validate that the proposed extensions drastically reduce the time required for reconciling ontology mappings for Case 1.

Figure 6.4: Detail comparison of the proposed extensions against Falcon [46], FOAM [22], Lily [104], AgreementMaker [19], and Prompt [76] on combination of 7 different data sets are given. This Figure shows the results for Case 1. Each graph i.e., A, B, C, D, E, and F shows the existing systems' results in comparison to the proposed extensions. Each graph consists of 5 pairs making 10 bars in total. Alternative pair is the results comparison of the proposed system against the existing system. 1^{st} bar of each pair shows the execution time (y-axis shows execution time) of the existing systems, while the 2^{nd} bar shows the execution time for the proposed extensions. Each packet of every bar (stacked column) in the graphs with different colors show the execution time consumed by the existing systems and the proposed extensions for reconciliation of mappings between various versions of the ontology. In these graphs; Hu=Human, Mo=Mouse, Fo=Food, He=Health, ACM=ACM, Sp=Springer, and PP=People+Pets are used as abbreviations for ontology names where V represents the version with number on it e.g., HuV2 represents Human ontology and its 2^{nd} version.

Case 2: As explained earlier, in this case, both ontologies evolved from one consistent state to another. Case 2 is also the worst case for the proposed system as mapping reconciliation procedure will look for changes in both ontologies and also execute the mapping reconciliation procedure for both ontologies. The existing systems are first checked for initial mappings between the ontologies, then for re-establishment of mapping using the changes in both ontologies. For mapping reconciliation, the existing systems with the proposed extensions are tested using the evolved ontologies. Both the existing systems and the proposed extensions are tested in detail using the data sets given in Table 6.4 with all their changes. The results of detail experiments for Case 2 are shown in Figure 6.5. Execution time shown in Figure 6.5 is all in minutes and fractions of minutes.

Each graph of Figure 6.5 represents the results of the existing systems and the proposed extensions on a particular data set with its different versions. Each graph of Figure 6.5 consists of 5 pairs making 10 bars in total. Each alternative pair is the results comparison of the proposed system against the existing system. 1^{st} bar of each pair shows the execution time of the existing system on each version (differentiated using colors) of the ontology, while the 2^{nd} bar of each pair shows the execution time for the proposed extensions for different versions of the ontology the same as Case 1. The detailed experimental results shown in Figure 6.5 validate the claims made in this research. This facilitates the process of interoperability and information exchange between web services. Thus the services are not suspended for longer durations due to evolving ontologies.

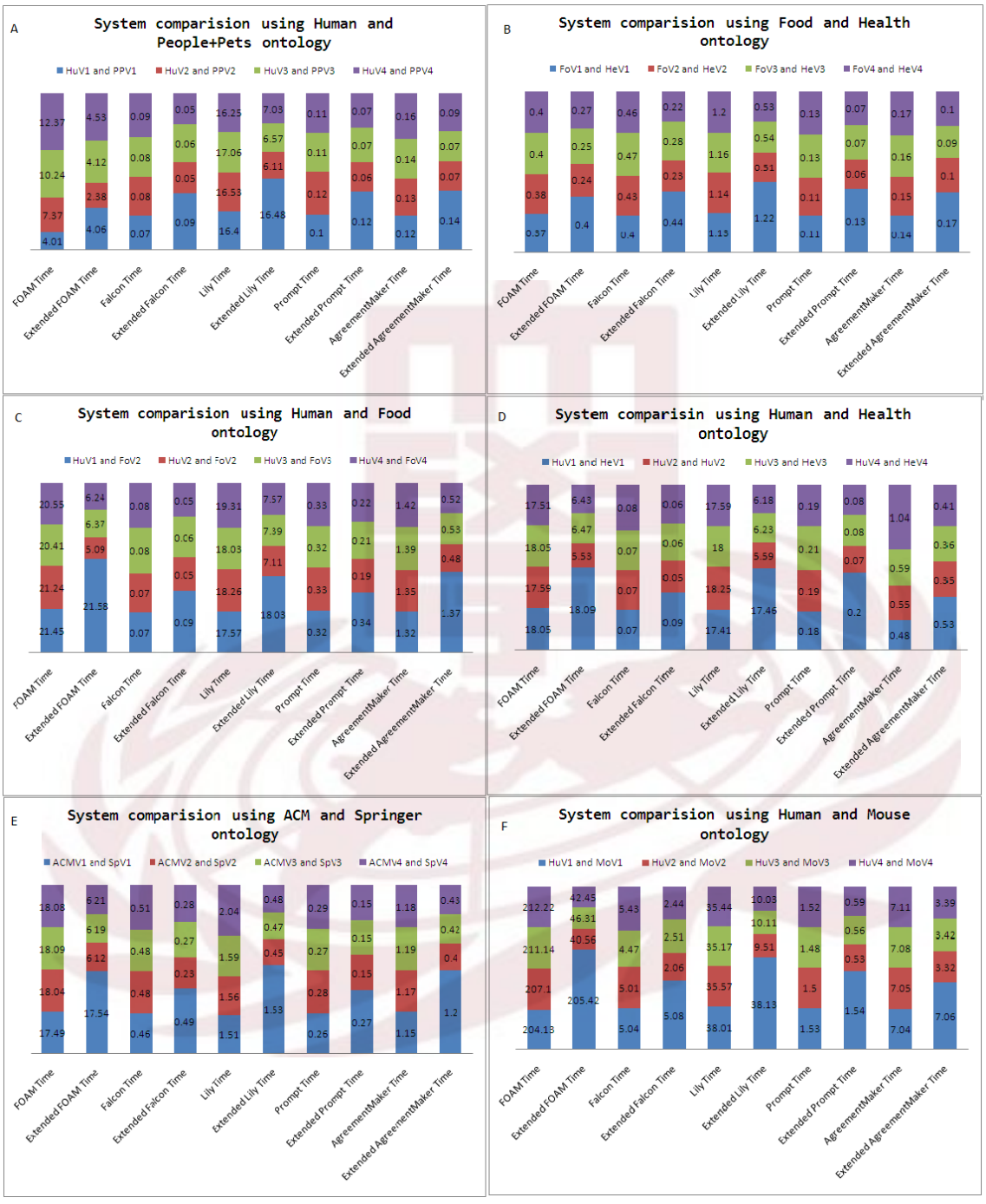
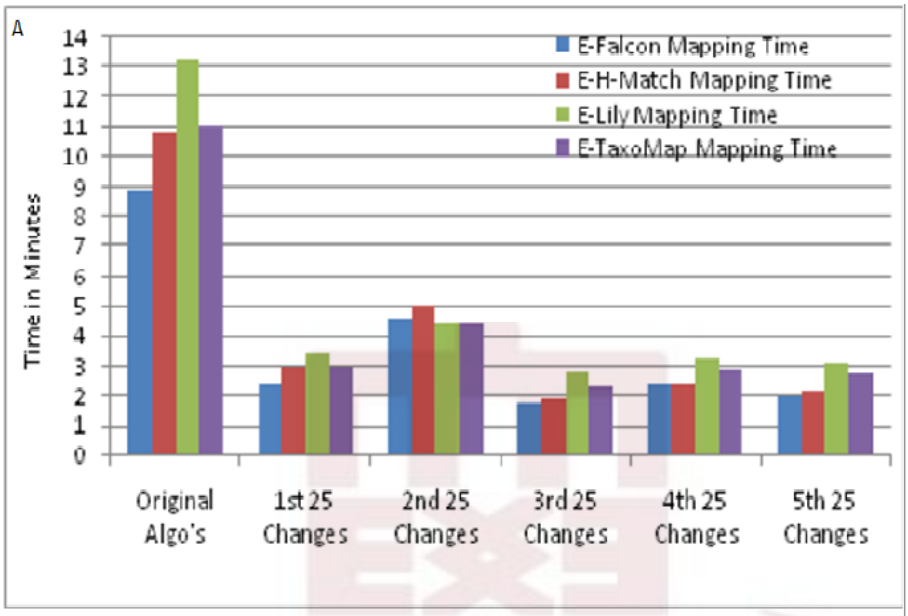


Figure 6.5: Detail comparison of the proposed extensions against Falcon [46], FOAM [22], Lily [104], AgreementMaker [19], and Prompt [76] on combination of 7 different data sets are given. This Figure shows the results for Case 2. Each graph i.e., A, B, C, D, E, and F shows the existing systems results in comparison to the proposed extensions. Each graph consists of 5 pairs making 10 bars in total. Each alternative pair is the results comparison of the proposed system against the existing system. 1^{st} bar of each pair shows the execution time (y-axis shows execution time) of the existing systems, while the 2^{nd} bar shows the execution time for the proposed extensions. Each packet of every bar (stacked column) in the graphs with different colors show the execution time consumed by the existing systems and the proposed extensions for reconciliation of mappings between various versions of the ontology. In these graphs; Hu=Human , Mo=Mouse, Fo=Food, He=Health, ACM=ACM, Sp=Springer, and PP=People+Pets are used as abbreviations for ontology names where V represents the version with number on it e.g., HuV2 represents Human ontology and its 2^{nd} version.

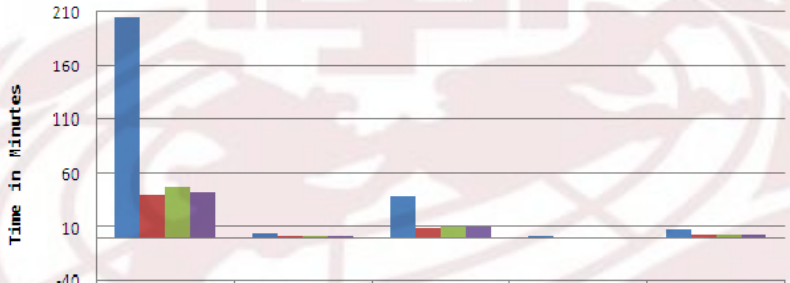
6.2.3 Effects of Change Type

The time for reconciliation of mappings between ontologies depends on the types of changes made. A single change introduced may have cascading effects on the existing resources or may result in several induced changes [27]. The proposed approach depends on number of changes. The more the number of changes in an ontology, the higher will be the mapping time for the proposed approach; however, still less than the original algorithms. Mostly, the cascade effects and induced changes are due to the change at higher level of hierarchy and are less frequent once domain ontology gets matured [27, 39]. One of such cases is also visible in Figure 6.6 (x-axis = no of tests, y-axis = minutes) in bar 3^{rd} comparison of Figure 6.6-A and Figure 6.6-B. Figure 6.6-A shows the results for complex type changes while Figure 6.6-B shows the results for atomic changes. 1^{st} bar in Figure 6.6-A and Figure 6.6-B is the original time of all the algorithms for establishing the mappings between Human and Mouse ontology while the remaining bars are the time results for the reconciliation of mapping with the proposed extensions using *CHL*. In Figure 6.6-A, set of 25 random changes (complex) are introduced to each version of the ontology. In Figure 6.6-B, changes (atomic) listed in Table 6.4 are introduced to each version of the ontology. In 3^{rd} bar combination of Figure 6.6-A and Figure 6.6-B, cascading effects make the reconciliation procedure take more time than the other reconciliation tests with the proposed extensions. Nevertheless, even with the cascade effects and induced changes, the proposed approach takes less mapping computation time than the original algorithms.



B

Impact of casecade changes on reconciliation procedure time



	Extended FOAM Time	Extended Falcon Time	Extended Lily Time	Extended Prompt Time	Extended AgreementMaker Time
HuV1 and MoV1	205.42	5.08	38.13	1.54	7.06
HuV2 and MoV2	40.56	2.06	9.51	0.53	3.32
HuV3 and MoV3	46.31	2.51	10.11	0.56	3.42
HuV4 and MoV4	42.45	2.44	10.03	0.59	3.39

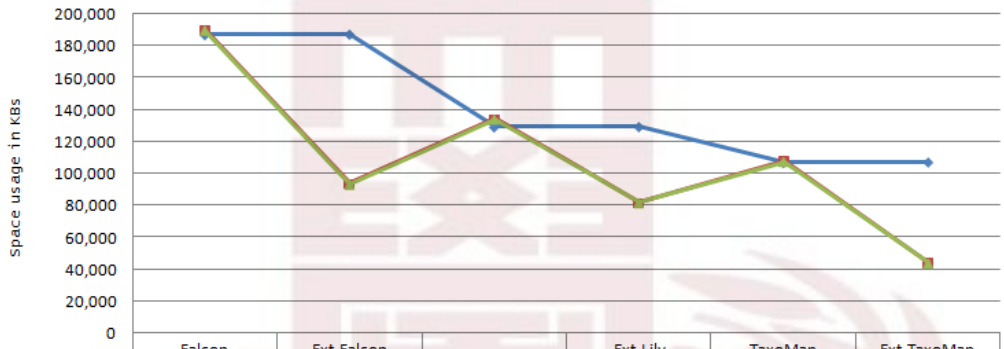
Figure 6.6: 6.6-A shows the mapping and re-establishment of mapping results for Mouse and Human ontology with complex changes. First bar combination is the result of original Falcon [46] and H-Match [17] while remaining bar combinations are the results of the proposed extension. The 3^{rd} bar shows the time increase due to the cascading effects of changes. 6.6-B shows the mapping and re-establishment of mapping results for Mouse and Human ontology with atomic changes. The 1^{st} bar combination is the result of original Falcon [46], FOAM [22], Lily [104], AgreementMaker [19], and Prompt [76] while remaining bar combinations are the results of the proposed extension. In 6.6-B, again the the 3^{rd} bar combination shows the time increase due to the cascading effects of changes. The same effects are also visible in the 3^{rd} row of the tabular view of results in Figure 6.6-B against the 2^{nd} and 4^{th} rows.

6.2.4 Memory Utilization

In addition to time efficiency of proposed algorithm, it has also proven to be space efficient. This is due to the fact that for mapping reconciliation procedure as against the existing systems, at any particular instant of time, loads ontology from one side and changes from another side. The changes are far lesser in size than the original ontology. The proposed systems' runtime memory usage is compared with the existing systems memory usage. The results (see Table 6.7) show that the proposed system memory consumption is lesser than the existing systems. Moreover, as the proposed system takes lesser time than the existing systems, so the memory consumption is also for shorter interval of time in comparison with the existing systems. Table 6.7 shows the results of memory consumption of proposed extensions to existing systems against the existing systems' traditional approach. *Human* and *Mouse* ontologies original versions and changed versions shown in Table 6.4 are used for this purpose. Efficient memory utilization using proposed extensions for mapping reconciliation in changed ontologies are highlighted in Table 6.7.

6.2.5 Reconciled Mapping Accuracy

Though, it is not the focus of this research; however, accuracy of generated mappings is an important issue. The proposed extensions reduce the amount of time required for the mapping reconciliation; however, it is also important to test its effects on the accuracy of reconciled mappings. In this section, detail results related to reconciled mapping accuracy is given (see Table 6.5). All the discussions in this section are based on atomic level changes and the accuracy of reconciled mappings with the help of atomic changes; the details of these atomic changes are given in Table 6.4. The results in Table 6.5 show the



	Falcon Runtime Space Usage	Ext-Falcon Runtime Space Usage	Lily Runtime Space Usage	Ext-Lily Runtime Space Usage	TaxoMap Runtime Space Usage	Ext-TaxoMap Runtime Space Usage
Human vs. Mouse (Original)	187,003	187,320	129,393	129,544	107,061	107,392
HumanV1 vs. MouseV1	189,934	93,698	134,319	81,802	107,801	44,061
HumanV2 vs. MouseV2	189,544	93,491	134,002	81,854	107,566	44,132

Figure 6.7: Space consumption analysis of Falcon [46], H-Match [17], Lily [104], and TaxoMap [41] against the systems with proposed extensions using *CHL* during the mapping reconciliation procedure. All the memory size is represented in KBs, whereas all the memory usage values are the peak memory usage values recorded during systems execution.

percentage (round off percentages are given) of overall mappings found after reconciliation procedure. The mappings found by the original mapping systems are considered as the total possible mappings while the mapping found with the proposed extensions are compared against the original mapping system's results. The details of data sets and the mapping systems that have been used for the experiments are also given in Table 6.5. During the logging process, each and every ontology change is logged in *CHL*, this also results in establishing/reconciling redundant mappings (that already exist in the original mappings) and are removed from the final list of reconciled mappings. The formula used to calculate the percentage is simple and is given below.

$$Percentage\ accuracy\ of\ reconciled\ mappings\ = \\ (No\ of\ reconciled\ mappings\ /\ No\ of\ original\ system\ mappings)\ *\ 100 \quad (6.1)$$

Most of the mapping systems developed are mainly focusing on the accuracy of the mappings and the accuracy of mapping is more critical when the services or information systems dealing with information from healthcare domain. To investigate about the accuracy of reconciled mappings, healthcare domain ontologies i.e., HL7 Classes ontology and openEHR Classes ontology have been used with their two different versions for mapping and mapping reconciliation. These ontologies have been tested using FOAM [22], Falcon [46], AgreementMaker [20, 19], and Lily [104] and their results are compared with results from proposed extensions to these systems (see Table 6.6). The changes used in these tests are also atomic changes and the numbers of changes introduced in different versions of the ontology are listed in Table 6.6. Like Table 6.5, it is also visible in Table 6.6 that the mappings found after reconciliation procedure is less than those found by

Ontology1	Ontology2	Changes	Ext-FOAM	Ext-Falcon	Ext-AgrMaker	Ext-Lily
HumanV1	MouseV1	Original	100%	100%	100%	100%
HumanV2	MouseV2	283 vs. 116	96.50%	96.50%	96.00%	96.00%
HumanV3	MouseV3	112 vs. 201	93.50%	95.00%	94.00%	95.00%
HumanV4	MouseV4	123 vs. 198	97.00%	97.00%	97.00%	97.00%
HumanV1	HealthV1	Original	100%	100%	100%	100%
HumanV2	HealthV2	283 vs. 169	97.00%	98.50%	98.00%	99.00%
HumanV3	HealthV3	112 vs. 153	96.50%	96.00%	97.00%	97.00%
HumanV4	HealthV4	123 vs. 145	99.00%	100%	100%	100%
HealthV1	FoodV1	Original	100%	100%	100%	100%
HealthV2	FoodV2	169 vs. 122	98.50%	100%	100%	100%
HealthV3	FoodV3	153 vs. 161	97.00%	98.50%	99.00%	99.00%
HealthV4	FoodV4	145 vs. 114	98.50%	99.50%	99.00%	100%
ACMV1	SpringerV1	Original	100%	100%	100%	100%
ACMV2	SpringerV2	109 vs. 176	99.00%	100%	100%	100%
ACMV3	SpringerV3	133 vs. 114	98.50%	99.50%	99.00%	99.50%
ACMV4	SpringerV4	141 vs. 106	99.50%	100%	100%	100%

Table 6.5: Shows mapping accuracy results of proposed extensions to the mapping systems against the original mapping systems. The results in this table are given for Human, Mouse, Health, Food, ACM, and Springer Ontologies and for the mapping process FOAM [22], Falcon [46], AgreementMaker [19], and Lily [104] are used. The results are not showing that the proposed extensions achieved 100% of mapping accuracy; however, these results shows the percentage of results (accuracy) achieved by proposed extensions in comparison to the original systems when the original systems re-establish the complete mappings.

the original systems. In addition, these tools also have some deficiencies in finding exact match for the concepts in the ontology. For instance, when HL7 Classes ontology using SNOMED CT (O_1) as a base line was matched with another HL7 Classes ontology using HL7 RIM (O_2) as its base model and $Event$ concept from O_2 is mapped with the $Event$ concept of O_1. However, they both have different semantics. The same way, $Event$ concept from O_2 have same semantics as $Clinical\_Findilgs$ in O_1; however, they are not matched by the above matching systems.

To overcome the decrease in accuracy of reconciled mappings, two points has been focused to work on. 1) The level of information with the changes has been increased. With every class change (except class deletion), extra information i.e., it's super class and sub classes has also been provided during reconciliation procedure. The same way with every property change (excluding property deletion), additional information of domain and range has been provided. Improvements have been found in the accuracy of reconciled mappings; however, this additional information has also introduced an increase in the mapping reconciliation time. 2) The reason is not only the additional information; there are also semantic conflicts that cannot be resolved without expert intervention as discussed above i.e., for $Event$ concept of *HL7 Classes* ontology. Currently, the focus is on knowing the missing mappings and reasons for the missing mappings that will help to optimize the proposed system for mapping accuracy as well.

Change capturing capability of proposed scheme is evaluated in this chapter. For this purpose, the proposed plug-in is also compared with 4 existing systems for their change capturing capability. Once the changes are logged in *CHL* then these are used for reconciliation of mappings. The process for mapping reconciliation using atomic and complex

Mapping Systems	**Ontology1**	**Ontology2**	**Ontology Changes**	**Mapping Time**	**No of Mappings Found**
FOAM	HL7V1	openEHRV1	Original	18.23 min	16
Ext-FOAM	HL7V1	openEHRV1	Original	18.57 min	16
FOAM	HL7V2	openEHRV2	103 vs. 166	19.30 min	19
Ext-FOAM	HL7V2	openEHRV2	103 vs. 166	4.03 min	17
Falcon	HL7V1	openEHRV1	Original	.58 min	18
Ext-Falcon	HL7V1	openEHRV1	Original	1.01 min	18
Falcon	HL7V2	openEHRV2	103 vs. 166	1.18 min	20
Ext-Falcon	HL7V2	openEHRV2	103 vs. 166	.26 min	19
AgrMaker	HL7V1	openEHRV1	Original	1.17 min	18
Ext-AgrMaker	HL7V1	openEHRV1	Original	1.24 min	18
AgrMaker	HL7V2	openEHRV2	103 vs. 166	1.49 min	20
Ext-AgrMaker	HL7V2	openEHRV2	103 vs. 166	.41 min	18
Lily	HL7V1	openEHRV1	Original	1.45 min	17
Ext-Lily	HL7V1	openEHRV1	Original	1.49 min	17
Lily	HL7V2	openEHRV2	103 vs. 166	2.09 min	19
Ext-Lily	HL7V2	openEHRV2	103 vs. 166	.49 min	18

Table 6.6: Shows mapping accuracy results of proposed extensions to the mapping systems against the original mapping systems using *HL7 Classes* Ontology and *openEHR Classes* Ontology. In these tests, only two versions of the said ontologies are used. For mapping process FOAM [22], Falcon [46], AgreementMaker [19], and Lily [104] are used.

changes is tested in this chapter. In addition, the tests are also conducted for both scenarios including change in one ontology as well as in both ontologies. The time consumption, memory consumption, and reconciled mappings accuracy is also comprehensively evaluated in this chapter. The next chapter provides the conclusions on the research area and the contributions of this research work in the area.

Chapter 7

Conclusion and Future Directions

This chapter concludes the research work, findings, and contributions of this thesis. The subsequent sections highlights the research area targeted in this research and the contributions made in this area. At the end, the potential future directions are discussed that can be worked on to extend this research work.

7.1 Conclusion

Ontologies are usually large, complex structured, and dynamic in nature. The issue with the complex and dynamic nature of domain ontologies is the lack of formal change representation scheme with comprehensible semantics. Changes in ontologies are incorporated to accommodate new knowledge; however, their effects suspends the usability of dependent applications. The uncontrolled, decentralized, and complex nature of dynamic web ontologies makes the ontology change management a complex and a collaborative task.

In this research, the problems of ontology change management are formulated, both philosophically and empirically. Different change representational schemes exist; however, they lack in formal and change principles based representation. The base research methods for ontology change representational scheme are principles of change and granularity level of change. These are the main focus areas of the proposed conceptualization, formalization, and representation of change. Based on the formalization and conceptualization, Change History Ontology (CHO) with basic constructs and details has been developed. It is the backbone representational model of the proposed ontology change management framework. It acts as a glue to bind different components in the framework and provide effective storage, retrieval, traceability, recovery, and mapping reconciliation services. CHO is used to record changes by creating a semantically structured Change History Log (CHL). The proposed system is developed as a plug-in for the ontology editor Protege to listen and log all the ontology changes. Moreover, it is configurable with any ontology editing tool that supports the hooks implemented in this plug-in. The developed plug-in listen to ontology changes and log them in *CHL*. The logged changes are later used for reconciliation of mappings between evolving web ontologies.

The change capturing ability of the proposed ontology change management framework developed as plug-in for Protege is compared with modern state of the art change capturing systems. The comparison results have demonstrated that the proposed framework has higher accuracy and better change coverage than the existing systems in terms of change capturing. The captured changes are later used for mapping reconciliation. The proposed mapping reconciliation algorithm is tested and evaluated exhaustively against the existing systems for performance, memory utilization, and mapping accuracy. All the tests and experiments are conducted using different versions of standard ontologies (data

sets) available online. It is found that the proposed extensions to the existing systems for mapping reconciliation have greatly improved the performance and memory utilization of existing systems. In addition, relatively good accuracy is maintained with the proposed extensions.

7.2 Contributions

7.2.1 Change History Ontology

Developing a representational structure for maintaining and managing ontology changes is a crucial task. The main contribution of this research is the development of such structure, logging ontology changes using the developed structure, and later using the logged changes for mapping reconciliation. To represent, maintain, and manage the ontology changes properly; a formally structured and semantically enrich ontological structure i.e., Change History Ontology (CHO) is developed in this research. Change History Log (CHL) uses *CHO* for storage, management, and effective retrieval of ontology changes.

7.2.2 Change Capturing

For the storage of ontology changes in *CHL*, change capturing is very important that can help in automatically log the changes in *CHL*. *Change Tracer* is implemented as a plug-in for Protege to capture ontology changes during its editing session and then log these changes in *CHL*. For the accuracy and validity of *Change Tracer*, it is tested extensively and compared with existing systems, such as *Change Tab*, *Version Log Generator*,

Change Detection, and *Change Capturing*. The information completeness for captured changes of *Change Tracer* is verified using ontology recovery procedure implementation.

7.2.3 Mapping Reconciliation

Information exchange and interoperability is the key research focus in different research groups. Mappings between two or more information sources (i.e., ontologies) of web services are the key for sharing information and achieving interoperability. However, due to the autonomous nature of services, and discovery of new knowledge in the field, the domain ontologies evolve which consequently make the existing mappings unreliable and stale. So mapping reconciliation is required to keep the services functioning for information exchange. In this research, extensions to existing mapping systems are proposed by introducing *CHL* that enables the functionality of mapping reconciliation in these systems, in time and memory efficient manner. It only consider the changed resources in ontologies for mapping reconciliation. The proposed extensions to existing systems are extensively evaluated using *FOAM*, *Falcon*, *H-Match*, *Prompt*, *Lily*, *AgreementMaker*, and *TaxoMap* on verity of different data sets. A drastic decrease in the amount of time and memory required for reconciliation of ontology mappings is found among dynamic ontologies compared to the existing systems that re-initiate the complete process. In addition, no significant variation in the mapping accuracy is found that can restrict the usability of proposed extensions.

7.3 Future Directions

Schema level changes are the main focus of this research; in addition, work on ontology changes at instance level can be a good extension to this work. This will involve the consistency checking and will result in reasoning jobs like validity and verifiability of instances. Moreover, the proposed scheme is extendable to collaborative environment which will make it usable for wider research and development community.

In the perspective of mapping reconciliation, variable mapping accuracy in results of the proposed technique is the main concerning point. This can also be a potential extension to the research work carried-out in this dissertation. To achieve the same level of accuracy, two schemes can be adopted.

1 The level of meta information with the changes can be increased which can help in achieving better accuracy. For example; with every class change (except class deletion), extra information i.e., it's super class and sub classes can also be provided during reconciliation procedure. This will also enforce the semantics of changed resources. The same way with every property change (excluding property deletion), additional information of domain and range can also be provided. Improvements can be found in accuracy of reconciled mappings using meta information; however, this additional information will also result in an increase to the mapping reconciliation time.

2 The reason for loss in accuracy is not only because of the limited meta information; there are also semantic conflicts that cannot be resolved without expert intervention as discussed in Chapter 6, Section 6.2.5 i.e., for $Event$ concept of *HL7 Classes* ontology. The work can be started by investigating the missing mappings and reasons

for the missing mappings. This will help to formulate a methodology for solving the mapping accuracy issue.

References

[1] C. E. Alchourrn, P. Grdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(1), 1985.

[2] Y. An and T. Topaloglou. Maintaining semantic mappings between database schemas and ontologies. In *Proceedings of Joint ODBIS and SWDB workshop on Semantic Web Ontologies Databases*, volume 5005/2008, Berlin, Germany, Feb 2008. Springer.

[3] K. Arkoudas and S. Bringsjord. Vivid: An ai framework for heterogeneous problem solving. *Artificial Intelligence*, 173(15), 2009.

[4] M. Ato, E. Ato, and J. Gmez. Analyzing change among developmental stages with categorical models. *Quality and Quantity*, 39(1), 2005.

[5] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *Proceedings of the ACM SIGMOD international conference on Management of data*, Baltimore, Maryland, USA, 2005. ACM.

[6] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *Int. Journal of Ad Hoc and Ubiquitous Computing*, 2(4), 2007.

[7] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. Oiled: A reasonable ontology editor for the semantic web. In *In Proceedings of the 24th German / 9th Austrian Conference on Artificial Intelligence (KI)*. LNCS.

[8] W. Behrendt, E. Gahleitner, K. Latif, A. Gruber, E. Weippl, S. Schaffert, and H. Kargl. Upper ontologies with specific consideration of dolce, sumo and sowas upper level ontology. Deliverable D121, DynamOnt Project, 2005.

[9] T. J. M. Bench-Capon and G. Malcolm. Formalising ontologies and their relations. In *Proceedings of the 10th int'l conference on Database and Expert Systems Applications (DEXA'99)*, Florence, Italy, 1999. LNCS, Springer.

[10] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5), 2001.

[11] P. B. Bhat, C. S. Raghavendra, and V. K. Prasanna. Efficient collective communication in distributed heterogeneous systems. In *Proceeding of 19th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Austin, TX, USA, June 1-4 1999. IEEE.

[12] E. Brynjolfsson and H. Mendelson. Information systems and the organization of modern enterprise. *Journal of Organizational Computing*, 3(3), 1993.

[13] S. C. Buraga, L. Cojocaru, and O. C. Nichifor. Survey on web ontology editing tools. *Transactions on Automatic Control and Computer Science*, 0(0), 2006.

[14] R. Buyyaa, 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*, 5(1), 2009.

[15] S. Castano, A. Ferrara, and G. N. Hess. Discovery-driven ontology evolution. In *3rd Italian Semantic Web Workshop: The Semantic Web Applications and Perspectives (SWAP)*, Pisa, Italy, December 18-20 2006. Deutsche Bibliothek.

[16] S. Castano, A. Ferrara, and S. Montanelli. Evolving open and independent ontologies. *Journal of Metadata, Semantics and Ontologies (IJMSO)*, 1(4), 2006.

[17] S. Castano, A. Ferrara, and S. Montanelli. Matching ontologies in open networked systems. *Techniques and applications, Journal on Data Semantics (JoDS)*, 3870/2006, 2006.

[18] 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*, 18(3), 2004.

[19] I. F. Cruz, F. P. Antonelli, and C. Stroe. Agreementmaker: Efficient matching for large real-world schemas and ontologies. *PVLDB*, 2(2), 2009.

[20] I. F. Cruz, W. Sunna, N. Makar, and S. Bathala. A visual tool for ontology alignment to enable geospatial interoperability. *Journal of Visual Languages and Computing*, 18(3), 2007.

[21] M. Doerr, Chair, and Heraklion. Crm, cidoc documentation standards working group.

[22] M. Ehrig and Y. Sure. Foam - framework for ontology alignment and mapping; results of the ontology alignment initiative. In *Proceedings of the Workshop on Integrating Ontologies*, volume 156. CEUR-WS.org, 2005.

[23] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 4th edition, 2003.

[24] J. Engelfriet and G. Rozenberg. *Node Replacement Graph Grammars, In Handbook of graph grammars and computing by graph transformation*. World Scientic Publishing Co, volume 1 edition, 1997.

[25] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. *Schema Mapping Evolution Through Composition and Inversion*. Springer-Verlag, Data-Centric Systems and Applications, Feb 2011.

[26] G. Flouris. On belief change in ontology evolution: Thesis. December 2006.

[27] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou. Ontology change: Classification and survey. *Knowledge Engineering Review (KER)*, 23(2), 2008.

[28] G. Flouris and D. Plexousakis. Handling ontology change:survey and proposal for a future research direction. Technical Report TR-362 FORTH-ICS, Institute of Computer Science, FORTH., Greece, 2005.

[29] G. Flouris, D. Plexousakis, and G. Antoniou. A classification of ontology changes. In *3rd Italian Semantic Web Workshop: Semantic Web Applications and Perspectives (SWAP) – Poster Session*, Pisa, Italy, December 18-20 2006. Deutsche Bibliothek.

[30] T. Gabel, Y. Sure, and J. Voelker. Kaon - ontology management infrastructure. Technical Report D3.1.1.a, SEKT Project: Semantically Enabled Knowledge Technologies, March 2004.

[31] E. Gahleitner, W. Behrendt, J. Palkoska, and E. Weippl. On cooperatively creating dynamic ontologies. In *Proceedings of the 16th ACM Conference on Hypertext and Hypermedia*, Salzburg, Austria, September 6-9 2005. ACM.

[32] E. Gahleitner, K. Latif, A. Gruber, and R. Westenthaler. Specification of methodology and workbench for dynamic ontology creation. Deliverable D201, DynamOnt Project, 2006.

[33] A. Gangemi. Ontology design patterns for semantic web content. In Y. Gil, E. Motta, R. Benjamins, and M. Musen, editors, *4th Intl Semantic Web Conf (ISWC)*, volume 3729, Galway, Ireland, November 6-10 2005. Springer.

[34] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc, San Francisco, USA, 1993.

[35] J. Guo. Using category theory to model software component dependencies. In *Proceedings of the 9th IEEE International Conference on Engineering of Computer-Based Systems (ECBS'02)*, Lund, Sweden, April 8-11 2002.

[36] T. R. Gurber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2), 1993.

[37] P. Haase and L. Stojanovic. Consistent evolution of owl ontologies. In *Proceedings of the 2nd European Semantic Web Conference (ESWC)*, Heraklion, Greece, May 29 - June 1 2005. LNCS.

[38] P. Haase and Y. Sure. State of the art on ontology evolution. Technical Report D3.1.1.b, SEKT Project: Semantically Enabled Knowledge Technologies, August 2004.

[39] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The piazza peer data management system. *IEEE Transactions on Knowledge and Data Engineering*.

[40] T. A. Halpin. *Information Modelling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufman Publishers, 1st edition, 2001.

[41] F. Hamdi, B. Safar, N. B. Niraula, and C. Reynaud. Taxomap alignment and refinement modules: Results for oaei 2010. In *In Proceedings of the 5th International Workshop on Ontology Matching (OM-2010) collocated with the 9th International Semantic Web Conference (ISWC)*, Shanghai, China, November 7-11 2010. LNCS.

[42] R. P. Hardie. *Physics by Aristotle*. eBooks@Adelaide, Adelaide University, http://etext.library.adelaide.edu.au/a/aristotle/a8ph/index.html edition, 2004.

[43] J. Hartmann, R. Palma, and Y. Sure. Omv - ontology metadata vocabulary. In *in: C. Welty (Ed.), ISWC 2005 - In Ontology Patterns for the Semantic Web*, volume 3729, Galway, Ireland, November 6-10 2005. Springer.

[44] 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.

[45] I. Horrocks. Ontology engineering: Tools and methodologies. tutorial in semantic days 07.

[46] W. Hu and Y. Qu. Falcon-ao: A practical ontology matching system. *Journal of Web Semantics*, 6(3), 2008.

[47] S. M. Huff, R. A. Rocha, B. E. Bray, H. R. Warner, and P. J. Haug. Anevent model of medical information representation. *JAMIA*, 2, 1995.

[48] J. ichi 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 (SWFAT)*, Nara, Japan, March 11-12 2003. Online.

[49] D. Jones, T. Bench-Capon, and P. Visser. Methodologies for ontology development. In J. Cuena, editor, *IFIP XV IT & KNOWS*, Budapest, Hungary, 1998. IFIP.

[50] S. Khan and P. Mott. Differential evaluation of continual queries. Technical Report 2001.11, School of Computing, the University of Leeds, May 2001.

[51] A. M. Khattak, K. Latif, M. Han, S. Lee, Y.-K. Lee, and H. I. Kim. Change tracer: Tracking changes in web ontologies. In *21st IEEE International Conference on Tools with Artificial Intelligence*, Newark, New Jersey, USA, November 2-4 2009. IEEE CS.

[52] A. M. Khattak, K. Latif, S. Khan, and N. Ahmed. Managing change history in web ontologies. In *Fourth International Conference on Semantics, Knowledge and Grid*, Beijing, China, December 3-5 2008. IEEE Computer Society.

[53] A. M. Khattak, K. Latif, S. Khan, and N. Ahmed. Ontology recovery and visualization. In *4th International Conference on Next Generation Web Services Practices*, Seoul, Korea, October 20-22 2008. IEEE Xplore.

[54] A. M. Khattak, K. Latif, S. Lee, and Y.-K. Lee. Ontology evolution: A survey and future challenges. In *The 2nd International Conference on u- and e- Service, Science and Technology (UNESST)*, Jeju, Korea, December 12-14 2009. LNCS.

[55] A. M. Khattak, K. Latif, S. Lee, Y.-K. Lee, and T. Rasheed. Building an integrated framework for ontology evolution management. In *12th International Conference on International Business Information Management Association*, Kulalampur, Malaysia, June 12-13 2009. IBIMA.

[56] A. M. Khattak, J. Mustafa, N. Ahmed, K. Latif, and S. Khan. Intelligent search in digital documents. In *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, Sydney, Australia, December 9-12 2008. IEEE Xplore/ACM/WIC.

[57] A. M. Khattak, Z. Pervez, K. Latif, A. M. J. Sarkar, S. Lee, and Y.-K. Lee. Reconciliation of ontology mappings to support robust service interoperability. In *In proceedings of 8th IEEE Conference on Service Computing (SCC 2011)*, Washington DC, USA, July 4-9 2011. IEEE CS.

[58] M. Klein. *Change Management for Distributed Ontologies*. Phd thesis, Vrije University, Netherlands, 2004.

[59] M. Klein, A. Kiryakov, D. Ognyanov, and D. Fensel. Finding and characterizing changes in ontologies. In *21st International Conference on Conceptual Modeling*, Tampere, Finland, October 7-11 2002. Springer-Verlag.

[60] M. Klein and N. Noy. A component-based framework for ontology evolution. In *IJCAI Workshop on Ontologies and Distributed Systems*, Acapulco, Mexico, August 9-10 2003. IJCAI.

[61] M. Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Madison, WI, USA, June 2-6 2002. ACM.

[62] Y. D. Liang. *Enabling Active Ontology Change Management within Semantic Web-based Applications*. Mini phd thesis, University of Southampton, 2006.

[63] Y. D. Liang, H. Alani, and N. Shadbolt. Ontology change management in protégé. In *AKT DTA Colloquium, Milton Keynes*, United Kingdom, December 2005. University of Southampton.

[64] C.-C. Lin and H.-C. Yen. A new force-directed graph drawing method based on edge-edge repulsion. *IEEE Computer Society*, 1(1), 2005.

[65] 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 (KR-06)*, Lake District, United Kingdom, June 2-5 2006. AAAI.

[66] W. Liu, T. Tudorache, and T. Redmond. Changes tab in protégé.

[67] A. Maedche, B. Motik, N. Silva, and R. Volz. Mafra - a mapping framework for distributed ontologies. In *In Proceedings of the 13th international Conference on Knowledge Engineering and Knowledge Management, ontologies and the Semantic Web*, London, UK, June 1-4 2002. Springer-Verlag.

[68] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. Ontologies for enterprise knowledge management. *IEEE Intelligent Systems*, 18(2), 2003.

[69] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, Srini-vasan, and K. Sycara. Bringing semantics to

web services :the owl-s approach. In *Presented at the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, SanDiego ,CA, USA, July 6 2004. Springer.

[70] B. A. Meza, C. Halaschek-Wiener, A. Sheth, and et al. Semantic web technology evaluation ontology (sweto), a nsf medium itr project.

[71] G. A. Miller. Wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4), 1990.

[72] R. Mizoguchi. Tutorial on ontological engineering: Part 3: Advanced course of ontological engineering. New Generation Comput, 2004.

[73] N. Noy, A. Chugh, W. Liu, and M. A. Musen. A framework for ontology evolution in collaborative environments. In *International Semantic Web Conference*, Athens, GA, USA, November 5-9 2006. LNCS.

[74] N. Noy and M. Klein. Ontology evolution: Not the same as schema evolution. *Knowledge and Information System*, 6(4), 2004.

[75] N. Noy, S. Kunnatur, M. Klein, and M. Musen. Tracking changes during ontology evolution. In *International Semantic Web Conference*, Sardinia, Italy, June 9-12 2002. LNCS.

[76] N. Noy and M. Musen. The prompt suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6), 2003.

[77] N. Noy and M. A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 450–455. AAAI Press, 2000.

[78] N. Noy and Team. Protg: http://protege.stanford.edu/.

[79] N. F. Noy, R. W. Fergerson, and M. A. Musen. The knowledge model of protege-2000: Combining interoperability and flexibility. In *In Proceedings of the 12th International Conferenceon Knowledge Engineering and Knowledge Management: Methods, Models, and Tools (EKAW)*, pages 17–32. Sringer-LNAI, 2000.

[80] D. Oberle, R. V. andB. Motik, and S. Staab. *In Handbook on Ontologies*, chapter An extensible ontology software environment. International Handbooks on Information Systems. Springer, 2004.

[81] 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.

[82] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *In proceedings of International Semantic Web Conference (ISWC)*, Sardinia, Italy, June 9-12 2002. LNCS.

[83] P. Plessers and O. D. Troyer. Ontology change detection using a versioning log. In *Proceedings of 4th International Semantic Web Conference (ISWC)*, Galway, Ireland, UK, November 6-10 2005. LNCS.

[84] P. Plessers, O. D. Troyer, and S. Casteleyn. Understanding ontology evolution: A change detection approach. *Web Semantics Science Services and Agents on the World Wide Web*, 5(1), 2007.

[85] C. Preist. A conceptual architecture for semantic web services. In *In 3rd International Semantic Web Conference (ISWC)*, Hiroshima, Japan, November 7-11 2004. LNCS.

[86] D. Rogozan and G. Paquette. Managing ontology changes on the semantic web. In *IEEE/WIC/ACM International Conference on Web Intelligence*, Compiegne, France, September 19-22 2005. IEEE CS/ACM/WIC.

[87] 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.

[88] A. Shaban-Nejad. *A Framework for Analyzing Changes in Healthcare Lexicons and Nomenclatures*. Phd thesis, Concordia University, Montreal, Quebec, Canada, April, 2010.

[89] P. Shvaiko and J. Euzenat. Ten challenges for ontology matching. In *Proceedings of the 7th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, Monterrey, Mexico, August 11-13 2008. LNCS.

[90] B. Smith. *Blackwell Guide to the Philosophy of Computing and Information*, chapter Ontology. Blackwell Philosophy Guides. Blackwell Publishing, October 2003.

[91] M. J. Smith, R. G. Dewar, K. Kowalczykiewicz, and D. Weiss. Towards automated change propagation; the value of traceability. Technical report, Heriot Watt University, http://www.macs.hw.ac.uk:8080/techreps/docs/files/HW-MACS-TR-0002.pdf, 2003.

[92] Sparqlpush: pubsubhubbub (push) interface for sparql endpoint.

[93] L. Stojanovic, A. Madche, B. Motik, and N. Stojanovic. User-driven ontology evolution management. In *European Conference on Knowledge Engineering and Management (EKAW)*, Siguenza, Spain, October 1-4 2002. LNCS.

[94] L. Stojanovic, A. Madche, N. Stojanovic, and R. Studer. Ontology evolution as reconfiguration-design problem solving. In *Proceedings of the 2nd International Conference on Knowledge Capture (K-CAP-03)*, Sanibel Island, FL, USA, October 23 - 25 2003. ACM.

[95] Y. Sure, J. Angele, and S. Staab. Ontoedit: Multi faceted inferencing for ontology engineering. *Journal on Data Semantics*, 1(1), 2003.

[96] Y. Sure, S. Bloehdorn, P. Haase, J. Hartmann, and D. Oberle. The swrc ontology - semantic web for research communities. In *In Proceedings of the 12th Portuguese Conference on Artificial Intelligence - Progress in Artificial Intelligence (EPIA)*, volume 3803 of *LNCS*, pages 218–231, Covilha, Portugal, December 5-8 2005. Springer.

[97] S. Tunnicliffe and I. Davis. Changeset. Online, 2005.

[98] M. Tury and M. Bielikova. An approach to detection ontology changes. In *First international workshop on adaptation and evolution in web systems engineering (AEWSE)*, Palo Alto, CA, USA, July 10-14 2006. ACM.

[99] M. Uschold. Building ontologies: Towards a unified methodology. In *16th Annual Conference of the British Computer Society Specialist Group on Expert Systems*, Cambridge, UK, September 1996. Online.

[100] M. VanAntwerp and G. Madey. Warehousing, mining, and querying open source versioning metadata. *Journal on Metadata Semantics*, 2008.

[101] A. Varzi. Change, temporal parts, and the argument from vagueness. *Dialectica*, 59(4), 2005.

[102] Y. Velegrakis, R. J. Miller, and L. Popa. Preserving mapping consistency under schema changes. *The VLDB Journal*, 13(3), 2004.

[103] L. T. Vinh, S. Lee, H. X. Le, H. Q. Ngo, H. I. Kim, M. Han, and Y.-K. Lee. Semi markov conditional random fields for accelerometer based activity recognition. *Journal of Applied Intelligence*, 35(2), 2010.

[104] 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.

[105] R. Wasserman. The problem of change. *Philosophy Compass*, 1(1), 2006.

[106] F. Zablith. Ontology evolution: A practical approach. In *Poster at Proceedings of Workshop on Matching and Meaning at Artificial Intelligence and Simulation of Behaviour (AISB)*, Edinburgh, UK, April 9 2009. Online.

Appendix A: Change Tracer Evaluation

A.1 Change Recovery

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

For the evaluation of the recovery procedure, two different versions of ontology i.e., O_{V1} and O_{V2} are used. The changes between the versions i.e., C_{Δ} were stored in Change History Log (*CHL*) using *CHO*. 35 different changes were manually incorporated in *Bibliography* ontology. All the changes were classified in three categories: (1) Hierarchy level changes, including the changes having effects on classes, properties, and their constraints, (2) Class level changes, changes resulting from modifications to classes and constraints on classes. These changes also contribute to changes at hierarchy level, (3) Property level changes, changes resulting from modifications to properties and constraints on the properties. They also contribute to changes at hierarchy level.

The number of hierarchy, class, and property changes were 10, 10, and 15 respectively. However, changes contributing from classes and properties made the number for hierarchy level changes bigger. After identifying and logging the changes between two versions, an equation for the verification of recovery procedure has been devised. As the proposed plug-in provides both $Rollback$ and $Rollforward$ facilities, so the equations for these procedures' verification are also separated.

Roll Back

To roll back the changes from O_{V2}, subtract all the changes i.e., C_Δ from the ontology that caused O_{V2} to evolve from O_{V1}. This subtraction of the changes from O_{V2} were all made using proposed recovery ($RollBack$) Algorithm 2. The equation for verification is as under;

$$\mathcal{O}_{Vx} \equiv \mathcal{O}_{V2} - \mathcal{C}_\Delta \tag{A.1}$$

$$difference\langle \mathcal{O}_{V1}, \mathcal{O}_{Vx} \rangle \equiv \emptyset \tag{A.2}$$

The recovery ($RollBack$ Algorithm) process was applied on O_{V2}. The recovered version was stored in another temporary version O_{Vx}. The temporary recovered version was checked against the available version O_{V1}. O_{V1} was differed from the recovered version i.e., O_{Vx} and if the difference was null (empty) then it means that the recovery process for roll back produced correct result.

Roll Forward

To roll forward the ontology from O_{V1}, add/apply all the changes i.e., C_Δ to the ontology that caused O_{V2} to evolve from O_{V1}. This addition of the changes to O_{V1} were all made using the

proposed recovery ($RollForward$) Algorithm 3. The equation for verification of $Rollforward$ algorithm is;

$$\mathcal{O}_{Vx} \equiv \mathcal{O}_{V1} + \mathcal{C}_{\Delta} \tag{A.3}$$

$$difference\langle \mathcal{O}_{V2}, \mathcal{O}_{Vx} \rangle \equiv \emptyset \tag{A.4}$$

The recovery ($RollForward$ Algorithm) process was applied on O_{V1}. The recovered version was stored in another temporary version O_{Vx}. The temporary recovered version was compared against the available version O_{V2}. Furthermore, O_{V2} was differed from the recovered version i.e., O_{Vx} and if the difference was found null (empty) then it means that the recovery process for roll forward produced correct result.

The difference between two ontology models was calculated using the $difference()$ method of $Model$ class from $JenaAPI$. Both versions are also checked using Prompt [77]. Using the *Bibliography* ontology, the $RollBack$ and $RollForward$ algorithms have been tested and got very good results. The details of these results are given in Table A.1, whereas their descriptions are given below.

For $RollBack$, the plug-in was tested 12 times and 5 correct results were obtained. The problems were: (1) when a $DomainAddition$ entry is rolled backed, it is reverted as $DomainDeletion$. So the algorithm actually has deleted the domain of some property; however, Protege internally assigns $owl : Thing$ as domain to all those properties which do not have any domain. (2) When datatype property is deleted, the range of that property is not captured properly. Because of these two problems, very low accuracy for $Rollback$ was recorded. These problems were solved and

the plug-in was tested for 12 more times. This time, 7 correct results were obtained. Only one issue was found, i.e., when a property is made as inverse property, the information about the other property to which this property is made inverse to, is missing. The domain issue was resolved by letting the domain of a property as empty, range problem by using the *difference()* method, and inverse property problem by introducing the *hasInverseTo* property in CHO. After the corrections, 12 more experiments were conducted and this time 12 correct results were obtained and no issues with the recovery procedure.

The $RollForward$ was implemented after the $RollBack$ was completely implemented and all the problems which were faced during $RollBack$ were removed. To justify that the system is correctly working for its $RollForward$ operation, the $RollForward$ operation was tested on all the 36 tests which were used to test the $RollBack$ operation. Out of 36 roll forward experiments, 36 correct results were obtained with 100% accuracy, as shown in Table A.1.

To validate that the system was not only tested on biased and controlled data sets, a detail system evaluation on four standard online available data sets is also provided. The details of all these experiments are given in the next section.

A.1.1 System Evaluation

In this section, detailed evaluation of the proposed recovery algorithms is presented. The proposed algorithms were tested with different versions of four different standard ontologies openly available. The reason for testing the system on four different data sets was to prove that the system is usable with varity of different ontologies and in uncontrolled envoirnments. Another reason for

the test was to cover as many aspects of ontology change as possible. As one can see in Table A.2 and A.3, in different ontologies the concentration of changes are different. For example, OMV and SWRC ontology have more changes from properties and axiom prospective, SWETO [70] has more on the class prospective, whereas, the CRM has mixed changes.

It is important to mention the process of logging the changes between different versions of ontologies. To log the changes in CHL, $ChangeTracer$ was configured with Protege and then the respective changes were performed to the ontologies to log them in CHL. Details of these changes, their types, their dependence, their most appropriate sequence, and effects were manually analyzed. For the confirmation of the change analysis between two versions of ontology, the completeness of the changes, and the correctness of the changes, *difference()* method of *Model* class from *Jena API* as well as the *Compare* operation of *Prompt* [77] algorithm were used. These logged changes were used for the evaluation of the system which is given below.

Evaluation using OMV

Ontology Metadata Vocabulary (OMV) (http://ontoware.org/frs/?group_id=39) [43] is used by the community for better understanding of the ontologies for the purpose of properly sharing and exchanging the information among organizations. To achieve this goal, this standard is set and agreed by the community for sharing and reusing of ontologies. OMV actually provides common set of terms and definitions describing ontologies, so called ontology metadata vocabulary. OMV have different versions available online containing different sets of concepts, properties, and restrictions. The developed plug-in has been tested on three different versions of OMV. The OMV versions used for the experimentation are omv-0.6.owl, omv-0.7.owl, and omv-0.91.owl.

Table A.2 shows complete details about the types and number of changes among different versions. These changes comprise class related, property related, and heirarchy related changes, which were captured and stored in *CHL* with the help of $ChangeTracer$. Using these logged changes, the $Rollback$ and $Rollforward$ procedures with the validity checking using $Rollback$ and $Rollforward$ techniques presented in Section A.1 and Section A.1 have been applied. All the recovered versions have been compared with the original version and they all are found error free.

Evaluation using SWRC Ontology

Semantic Web for Research Communities (SWRC) ontology [96], models key entities relevant for research communities and related concepts in the domain of research and development. Currently there are 70 different concepts with 48 object type properties and 46 datatype properties. The reuse of ontologies for the real realization of semantic web and its continues improvement by user communities is a crucial aspect. The description and the usage guidelines are provided for SWRC ontology by the authors to make a complete value of the implicit and explicit facilities.

Two versions of SWRC ontology available online have been used, which are swrc-v0.3.owl and swrc-updated-v0.7.1.owl. Changes between version swrc-v0.3.owl and swrc-updated-v0.7.1.owl comprise class related, property related, and hierarchy related changes, which were captured and stored in *CHL* with the help of $ChangeTracer$. Details are given in the Table A.2. Using these logged changes, the $Rollback$ and $Rollforward$ techniques presented in Section A.1 and Section A.1 have been applied, and that resulted in recovered versions. All the recovered versions have

been compared with the original version and they all are found correct.

Evaluation using CRM Ontology

One of the ontologies used for experiments is the CIDOC Conceptual Reference Model (CRM) [21]. CRM provides a common language and semantic framework for experts and developers in the cultural heritage domain and facilitates in sharing the understanding of cultural heritage information. Multiple versions of CRM are available online . For the evaluation of the proposed system, two different versions of CRM ontology (i.e., cidoc-crm-3.2.rdf and cidoc-crm-3.4.rdf) are used.

Changes between cidoc-crm-3.2.rdf and cidoc-crm-3.4.rdf were first captured and stored in *CHL* with the help of $ChangeTracer$. Details of all these changes are given in Table A.3. Using these logged changes, the $Rollback$ and $Rollforward$ techniques presented in Section A.1 and Section A.1 have been applied. The $Rollback$ from cidoc-crm-3.4.rdf with the help of logged changes produced a temporary recovered version. The recovered version has been checked against cidoc-crm-3.2.rdf, both versions are found the same. The $Rollforward$ from cidoc-crm-3.2.rdf with the help of logged changes produced a temporary recovered version. The recovered version has been compared against cidoc-crm-3.4.rdf, both versions are found the same with no differences between them. This shows that the $RollBack$ and $RollForward$ operations on cidoc-crm-3.2.rdf and cidoc-crm-3.4.rdf are correct.

Evaluation using SWETO

Semantic Web Technology Evaluation Ontology (SWETO) is basically an ontology developed as a benchmark by Semantic Web Community for evaluating the scalability of the available semantic web tools. More details on SWETO can be found on [70]. Since it is a benchmark ontology for testing the performance and scalability of semantic web tools, that's why it has been selected it to test the scalability and performance of the developed plug-in. Currently, three versions of the SWETO ontology [70] with names: (i) testbed-v1-4.owl (ii) testbed-v1-3.owl, and (iii) testbed-v1-2.owl are used for the experiments.

Since three versions of ontology are used, so two set of experiments are conducted. Initially, the first two versions, i.e., testbed-v1-2.owl and testbed-v1-3.owl are used. All the changes between these versions are stored in their corresponding log file. The details of these changes are given in Table 4. Using these logged changes, the $Rollback$ and $Roll forward$ techniques presented in Section A.1 and Section A.1 have been applied. The $Rollback$ and $Roll forward$ operations on testbed-v1-3.owl and testbed-v1-2.owl are implemented in the same way as discussed in previous experiments. With this implementation and comparison, all the results observed are found correct. This shows that the $Rollback$ and $Roll forward$ operations on the testbed-v1-2.owl and testbed-v1-3.owl are correct. Secondly, the other set of its two versions i.e., testbed-v1-3.owl and testbed-v1-4.owl are used for the experiments. All the changes between these two versions are also stored in CHL. The number and types of changes are given in Table A.3. Using these logged changes, the $Rollback$ and $RollForward$ techniques presented in Section A.1 and Section A.1 have been applied. The $Rollback$ and $Roll forward$ operations are applied in the same way as in the previous steps. The differences of original versions and recovered versions from $RollBack$ and $RollForward$ are analyzed for verification. The results of all the differences are

empty which shows that the $Rollback$ and $Roll forward$ operations on the testbed-v1-3.owl and testbed-v1-4.owl are correct.

(Rollback or Undo Algorithm) This algorithm assumes a pre-defined function, **TimeIndexedSort** for sorting member entries of the $ChangeSet$ based on their timestamp.
Input: An ontology $\mathcal{O}$.
Input: An instance of $ChangeSet$, $\mathcal{S}_\Delta \in ChangeSet$, which lists the changes made in the ontology $\mathcal{O}$.
Output: The previous version $\mathcal{O}'$ of the ontology $\mathcal{O}$ after reverting the changes mentioned in $\mathcal{S}_\Delta$.
1. /* Sort member entries of the $ChangeSet$ in descending order of their timestamp*/
2. TimeIndexedSort($\mathcal{S}_\Delta$, 'DESC')
3. **foreach** $c_\Delta \in \mathcal{S}_\Delta$ **do**
4. /* Process resource addition */
5. **if** $c_\Delta : OntologyChange \sqcap \exists changeType.Create$ **then**
6. /* Remove the resource(s) which were target of the change */
7. $\mathcal{O} \leftarrow \mathcal{O} - \{x | \langle c_\Delta, x \rangle changeTarget\}$
8. **else**
9. /* Process resource deletion */
10. **if** $c_\Delta : OntologyChange \sqcap \exists changeType.Delete$ **then**
11. $\mathcal{O} \leftarrow \mathcal{O} + \{x | \langle c_\Delta, x \rangle changeTarget\}$
12. **else**
13. /* Process modification*/
14. ...
15. /* Implementation of this algorithm consists of a number of other conditional statements to check the change type and to process it accordingly, such as for annotations.*/
16. **endif**
17. **end**

(RollForward or Redo Algorithm) This algorithm assumes a pre-defined function, **TimeIndexedSort** for sorting member entries of the $ChangeSet$ based on their timestamp.
Input: An ontology $\mathcal{O}$.
Input: An instance of $ChangeSet$, $\mathcal{S}_\Delta \in ChangeSet$, which lists the changes made in the ontology $\mathcal{O}$.
Output: The next version $\mathcal{O}'$ of the ontology $\mathcal{O}$ after re-implementing the extracted changes from CHL mentioned in $\mathcal{S}_\Delta$.
1. /* Sort member entries of the change set in ascending order of their timestamp and select the most recent one*/
2. TimeIndexedSort($\mathcal{S}_\Delta$, 'ASC')
3. **foreach** $c_\Delta \in \mathcal{S}_\Delta$ **do**
4. /* Process resource addition */
5. **if** $c_\Delta : OntologyChange \sqcap \exists changeType.Create$ **then**
6. /* Re-implement (insert) the resource(s) which were target of the change */
7. $\mathcal{O} \leftarrow \mathcal{O} + \{x | \langle c_\Delta, x \rangle changeTarget\}$
8. **else**
9. /* Process resource deletion */
10. **if** $c_\Delta : OntologyChange \sqcap \exists changeType.Delete$ **then**
12. /* Re-implement (delete) the resource(s) which were target of the change */
11. $\mathcal{O} \leftarrow \mathcal{O} - \{x | \langle c_\Delta, x \rangle changeTarget\}$
12. **else**
13. /* Process modification*/
14. ...
15. /* Implementation of this algorithm also consists of a number of other conditional statements to check the change type and to process it accordingly.*/
16. **endif**
17. **end**

Roll Back				
Details	**Tests**	**Correct Results**	**Problems**	**Accuracy**
Initial Attempts:	12	5	Domain Addition, Datatype Property Range Deletion	41.67
First Revision:	12	7	Inverse Property	58.34
Second Revision:	12	12	Nil	100
Roll Forward				
Details	**Tests**	**Correct Results**	**Problems**	**Accuracy**
Total Attempts:	36	36	Nil	100

Table A.1: Change logging validation by implementing Roll Back and Roll Forward.

Ontology Versions	OMV.owl & OMV-0.7.owl	OMV-0.7.owl & OMV-0.91.owl	swrc-v0.3.owl & swrc-updated-v0.7.1.owl
Total Changes	38	189	310
Change in Hierarchy	18	71	131
Change in Classes	6	34	84
Change in Properties	25	123	172

Table A.2: Roll Back and Roll Forward procedures' results for OMV and SWRC Ontologies.

Ontology Versions	cidoc-crm-3.2.rdf & cidoc-crm-3.4.rdf	testbed-v1-2.owl & testbed-v1-3.owl	testbed-v1-3.owl & testbed-v1-4.owl
Total Changes	170	124	223
Change in Hierarchy	94	60	170
Change in Classes	54	107	193
Change in Properties	103	14	22

Table A.3: Roll Back and Roll Forward procedures' results for CIDOC-CRM and SWETO Ontologies.

Appendix B: List of Publications

B.1 International Journal Papers

1 **Asad Masood Khattak**, Khalid Latif, and Sungyoung Lee, "Change Management in Evolving Web Ontologies", Knowledge-based Systems, (IF: 1.574), 2012, (In Press).

2 **Asad Masood Khattak**, Zeeshan Pervez, Khalid Latif, and Sungyoung Lee, "Time Efficient Reconciliation of Mappings in Dynamic Web Ontologies", Journal of Knowledge-based Systems, (IF: 1.574), 2012, (In Press).

3 **Asad Masood Khattak**, Zeeshan Pervez, Khalid Latif, and Sungyoung Lee, "Change History Ontology: A Theoretical Perspective", Journal of Advance Science Letters (IF: 1.253), ISSN:1936-6612, 2012

4 **Asad Masood Khattak**, Zeeshan Pervez, Wajahat Ali Khan, Sungyoung Lee and Eui-Nam John Huh, "Automatic System for Rule Learning and Evolution", Journal of Advance Science Letters (IF: 1.253), ISSN:1936-6612, 2012.

5 Zeeshan Pervez, **Asad Masood Khattak**, Sungyoung Lee and Christopher Nugent, "Privacy Aware Searching in Cloud Storage with Oblivious Index Based Data Search", Ad-

vanced Science Letters (IF: 1.28), 2012

6 **Asad Masood Khattak**, Phan Tran Ho Truc, Le Xuan Hung, La The Vinh, Viet-hung Dang, Donghai Guan, Zeeshan Pervez, Manhyung Han, Sungyoung Lee and Young-koo Lee, "Towards Smart Homes Using Low Level Sensory Data", Journal of Sensors (IF 1.77), ISSN: 1424-8220, 2011.

7 Zeeshan Pervez, **Asad Masood Khattak**, Sungyoung Lee and Young-Koo Lee, "SAPDS: Self-Healing Attribute-Based Privacy Aware Data Sharing in Cloud", Journal of Supercomputing (IF 0.534), pp.11581-11604, ISSN: 0920-8542, January 7, 2011.

8 **Asad Masood Khattak**, Zeeshan Pervez, Sungyoung Lee and Young-Koo Lee, "Intelligent Healthcare Service Provisioning using Ontology with Low Level Sensory Data", Transaction on Internet Information Systems (TIIS) (IF: 0.164) ISSN: 1976-7277, Vol.5, No 11, pp. 2016-2034, 2011.

9 Zeeshan Pervez, **Asad Masood Khattak**, Sungyoung Lee, Young-Koo Lee, "Achieving Dynamic and Distributed Session Management with Chord for Software as a Service Cloud", Journal of Software (JSW, ISSN 1796-217X), Academy Publisher, 2011.

10 **Asad Masood Khattak**, A.M. Khan, Sungyoung Lee and Young-Koo Lee, "Analyzing Association Rule Mining and Clustering on Sales Day Data with XLMiner and Weka", International Journal of Database Theory and Application , April, 2010.

B.2 International Journal Papers Under Review

1 **Asad Masood Khattak**, Zeeshan Pervez, Khalid Latif, Sungyoung Lee, Young-Koo Lee, ”Ontology Convergence after Evolution”, Journal of Information Science and Engineering, (IF 0.31), 2012. (Under Review)

B.3 International Conference Papers

1 **Asad Masood Khattak**, Zeeshan Pervez, Manhyoung Han, Sumgyoung Lee and Chris Nugent, ”DDSS: Dynamic Decision Support System for Elderly”, The 25th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2012), Rome, Italy, June 20-22, 2012

2 Wajahat Ali Khan, Maqbool Hussain, **Asad Masood Khattak**, Bilal Amin, and Sungyoung Lee, Integration of HL7 Compliant Smart Home Healthcare System and HMIS, 10th International Conference On Smart homes and health Telematics, 12-15, Italy, June, 2012, (Accepted)

3 Wajahat Ali Khan, Maqbool Hussain, **Asad Masood Khattak**, Muhammad Bilal Amin, Sungyoung Lee, SaaS based Interoperability Service for Semantic Mappings among Healthcare Standards The 8th International Conference on Innovations in Information Technology, UAE, March 18-20, 2012.

4 Wajahat Ali Khan, Maqbool Hussain, **Asad Masood Khattak**, Muhammad Bilal Amin, Sungyoung Lee, Achieving Interoperability among Healthcare Standards: Building Semantic Mappings at Models Level, In Proceedings ICUIMC, Malaysia, Febuary 20-22, 2012.

5 **Asad Masood Khattak**, Zeeshan Pervez, Wajahat Ali Khan, Sungyoung Lee, and Young-Koo Lee, "A Self Evolutionary Rule-base", The 4th International Conference on u - and e - service, Science and Technology (UNESST'11), Jeju, Korea, December 8 10, 2011.

6 **Asad Masood Khattak**, Zeeshan Pervez, Khalid Latif, A. M. Jehad Sarkar, Sungyoung Lee and Young-Koo Lee, "Reconciliation of Ontology Mappings to Support Robust Service Interoperability", The 8th IEEE International Conference on Services Computing (IEEE SCC 2011), Washington DC, July 4-9, 2011.

7 **Asad Masood Khattak**, Zeeshan Pervez, Sungyoung Lee and Young-Koo Lee, "Activity Manipulation using Ontological Data for u-Healthcare", The 8th International Conference on Wearable Micro and Nano Technologies for Personalized Health (pHealth 2011), Lyon, France, June 29 - July 1, 2011.

8 Iram Fatima, **Asad Masood Khattak**, Young-Koo Lee and Sungyoung Lee, "Automatic Documents Annotation by Keyphrase Extraction in Digital Libraries using Taxonomy", FutureTech 2011 Conference, Crete, Greece, June 28-30, 2011.

9 **Asad Masood Khattak**, Khalid Latif, Zeeshan Pervez, Iram Fatima, Sungyoung Lee and Young-Koo Lee, "Change Tracer: A Protg Plug-in for Ontology Recovery and Visualization", The 13th Asia-Pacific Web Conference (APWeb2011), (LNCS Conference), Beijing, China, April 18-20, 2011.

10 Zeeshan Pervez, **Asad Masood Khattak**, Sungyoung Lee and Young-Koo Lee, "CSMC: Chord based Session Management Framework for Software as a Service Cloud", The 5th ACM International Conference on Ubiquitous Information Management and Communication (ACM ICUIMC 2011), Seoul, Korea, February 21-23, 2011

11 **Asad Masood Khattak**, Zeeshan Pervez, Iram Fatima, Sungyoung Lee, Young-Koo Lee, ”Towards Efficient Analysis of Activities in Chronic Disease Patients”, The 7th International Conference on Ubiquitous Healthcare, Jeju, Korea, October 2010.

12 A. M. Jehad Sarkar, Adil Mehmood Khan, **Asad Masood Khattak**, S. K. Tanbeer, Young-Koo Lee, Sungyoung Lee, ”WAST: Web-Based Activity Sampling Tool for Activity Recognition”, In proceedings of the 2nd International Conference on Emerging Database (EDB 2010), pp. 178-183, Jeju, Korea, August 30-31, 2010.

13 **Asad Masood Khattak**, Zeeshan Pervez, Koo Kyo Ho, Sungyoung Lee, Young-Koo Lee, ”Intelligent Manipulation of Human Activities using Cloud Computing for u-Life Care”, The 10th Annual International Symposium on Applications and the Internet (SAINT 2010)”, Seoul, Korea, July 2010.

14 **Asad Masood Khattak**, Zeeshan Pervez, Jehad Sarkar, Young-Koo Lee, ”Service Level Semantic Interoperability”, International Workshop on Computing Technologies and Business Strategies for u-Healthcare (CBuH 2010)”, Seoul, Korea, July 2010.

15 **Asad Masood Khattak**, La The Vinh, Dang Viet Hung, Phan Tran Ho Truc, Le Xuan Hung, D. Guan, Zeeshan Pervez, Manhyung Han, Sungyoung Lee, Young-Koo Lee, ”Context-aware Human Activity Recognition and Decision Making”, ”12th International Conference on e-Health Networking, Application Services”, Lyon, France, July, 2010.

16 **Asad Masood Khattak**, Zeeshan Pervez, Sung-young Lee, Young-Koo Lee, ”After Effects of Ontology Evolution”, The 5th International Conference on Future Information Technology (FutureTech10), Busan Korea, May, 2010.

17 Zeeshan Pervez, **Asad Masood Khattak**, Sungyoung Lee and Young-Koo Lee, ”Dual Validation Framework for Multi-Tenant SaaS Architecture”, IEEE 5th International Conference

on Future Information Technology, http://www.ftrg.org/futuretech2010, Busan, Korea, May 21-23, 2010.

18 Xuan Hung Le; Sung-young Lee; Phan Truc; La The Vinh; **Asad Masood Khattak**, Manhyung Han; Dang Viet Hung; Hassan, M.M.; Kim, M.; Kyo-Ho Koo; Young-Koo Lee; Eui-Nam Huh; "Secured WSN-Integrated Cloud Computing for u-Life Care",7th IEEE Consumer Communications and Networking Conference (CCNC), 2010

19 **Asad Masood Khattak**, Khalid Latif, Sung-young Lee, Young-Koo 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.

20 **Asad Masood Khattak**, Adil Mehmood Khan, Tahir Rasheed, Sung-young Lee, Young-Koo Lee, "Comparative Analysis of XLMiner and Weka for Association Rule Mining and Clustering", The 2009 International Conference on Database Theory and Application (DTA 09), Jeju, Korea, December 10 12, 2009.

21 **Asad Masood Khattak**, Khalid Latif, Sung-young Lee, Young-Koo Lee, Manhuyng Han, and Hyoung Il Kim, "Change Tracer: Tracking Changes in Web Ontologies",21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Newark, USA, November 2009.

22 **Asad Masood Khattak**, Khalid Latif, Sung-young Lee, Young-Koo Lee, and Tahir Rasheed, "Building an Integrated Framework for Ontology Evolution Management",12th Conference on Creating Global Economies through Innovation and Knowledge Management: Theory and Practice (IBIMA), Malaysia, pp. 55-60, 29-30 June 2009.

B.4 Patents

1 **Asad Masood Khattak**, Sungyoung Lee, Young-Koo Lee, Hyoung Il Kim and Manhyung Han, ”Method for Reconciling Mappings in Dynamic/Evolving Web-Ontologies Using Change History Ontology”, International Patent No. 12/576,342, Oct. 9, 2009.

2 Sungyoung Lee, **Asad Masood Khattak**, ”Data Processing Method And Apparatus For Clinical Decision Support System. USA Patent, (2012). (Pending).