Change Impact Analysis for Evolving Ontology-based Content Management

Yalemisew Mintsnote Abgaz

BSc., MSc.

A Dissertation submitted in fulfilment of the

requirements for the award of

Doctor of Philosophy (Ph.D.)

to the

DCU

Dublin City University

Faculty of Engineering and Computing,

School of Computing

Supervisor: Dr. Claus Pahl

January, 2013

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Yalemisew Mintsnote Abgaz

Student ID: 58116745

Date: January, 2013

Examiners:

Dr. Paolo Ceravolo

Department of Information Technology

University of Milan

Italy

Dr. Gareth Jones

Faculty of Engineering and Computing

School of Computing

Dublin City University

Ireland

Dedication

To

My parents

Acknowledgements

The things I explore, the knowledge I amass and the secrets I fathom, the time I invest and the laziness I enjoyed, the mistakes I commit and the things I get them right or wrong relate to someone who have come to my life. There is someone who is with me throughout my pursuit of knowledge. Someone shines on my way, and I see things vividly. My deepest gratitude goes to God. Dear God, as your word says, all things come into being through you and apart from you not one thing came into being (John1:3). Thank you Jesus for helping me in my entire endeavour.

I always appreciate the invaluable support I received from my supervisor. It is my great experience to work with you Claus. You are a superb and paragon supervisor with great patience and persistence. My extravagant appreciation is directed to you. Thank you for everything. I would like to extend my deep gratitude to Dr. Gareth Jones and Dr. Paolo Ceravolo for your willingness to examine my work and give me feedbacks.

Would I forget your valuable inputs in my life? My mom and my dad, thank you for standing beside me all the time. My brothers and sisters, your prayer has lifted me up to this point. Thank you so much. Zemenu and Meklit, thank you for taking care of the family in my absence. Beza, I always value your input for my research and for my life during this study. I would take this opportunity to thank you and acknowledge that your presence really made a difference during my study.

I have received a lot from my colleagues in DCU, Javed, your inputs were critical and made me think critically. You are a hard-working colleague and charged me with a positive charge towards my work. AAkash and Pooyan, thank you for your endless collaborations and mind boggling and eye opening discussions. Thank you Ewnetu for your comments and discussions. My whole journey in DCU is accompanied by CNGLers, Thank you all. My special thank goes to Eithne McCann, for your magnanimity.

Finally, thank you all my friends from the church. Your presence made me enjoy my stay in Ireland. Lidya and Mitin, thank you for making my early days enjoyable. Ana and Markella, I always remember the comfortable environment we had after school hours.

Abstract

Ontologies have become ubiquitous tools to embed semantics into content and applications on the semantic web. They are used to define concepts in a domain and allow us to reach at a common understanding on subjects of interest. Ontologies cover wide range of topics enabling both humans and machines to understand meanings and to reason in different contexts. They cover topics such as semantic web, artificial intelligence, information retrieval, machine translation, software development, content management, etc. We use ontologies for semantic annotation of content to facilitate understandability of the content by humans and machines. However, building ontology and annotations is often a manual process which is error prone and time consuming.

Ontologies and ontology-driven content management systems (OCMS) evolve due to a change in conceptualization, the representation or the specification of the domain knowledge. These changes are often immense and frequent. Implementing the changes and adapting the OCMS accordingly require a huge effort. This is due to complex impacts of the changes on the ontologies, the content and dependent applications. Thus, evolving the OCMS with minimum and predictable impacts is among the top priorities of evolution in OCMS.

We approach the problem of evolution by proposing a framework which clearly represents the interactions of the components of an OCMS. We proposed a layered OCMS framework which contains an ontology layer, content layer and annotation layer. Further, we propose a novel approach for analysing impacts of change operations. Impacts of atomic change operations are assigned individually by analysing the target entity and all the other entities that are structurally or semantically dependent on it. Impacts of composite change operations are analysed following three stage process. We use impact cancellation, impact balancing and impact transformation to analyse the impacts when two or more atomic changes are executed as part of a composite or domain-specific change operation.

We build a model which estimates the impacts of a complete change operation enabling the ontology engineer to specify the weight associated with each optimization criteria. Finally, the model identifies the implementation strategy with minimum cost of evolution. We evaluate our system by building a prototype as a proof of concept and find out encouraging results.

Contents

List of Tables

List of Figures

List of Publication

- Abgaz, Y., Javed, M., and Pahl, C. (2012). Analysing impacts of change operations in evolving ontologies. Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn), collocated at ISWC2012. Boston, USA. 12th November, 2012.

- Javed, M., Abgaz, Y., and Pahl, C. (2012). Composite Ontology Change Operators and their Customizable Evolution Strategies. Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn), collocated at ISWC2012. Boston, USA. 12th November, 2012.

- Abgaz, Y., Javed, M., and Pahl, C. (2012). Dependency analysis in ontology-driven content-based systems. In L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. Zadeh, and J. Zurada (Eds.), Artificial Intelligence and Soft Computing, volume 7268 of Lecture Notes in Computer Science (pp. 3 - 12).

- Abgaz, Y., Javed, M., and Pahl, C. (2011). A framework for change impact analysis of ontology-driven content-based systems. In On the Move to Meaningful Internet Systems: OTM 2011 Workshops, Lecture Notes in Computer Science.

- Javed, M., Abgaz.Y. and Pahl. C. (2011). Graph-based discovery of ontology change patterns. In ISWC Workshops: Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn), 24th October, 2011, Bonn, Germany.

- Javed, M., Abgaz, Y. M., and Pahl, C. (2011). Towards implicit knowledge discovery from ontology change log data. In Knowledge Science, Engineering and Management (pp. 136 - 147).

- Javed, M., Abgaz, Y., and Pahl, C. (2011). A layered framework for pattern-based ontology evolution. In 3rd International Workshop Ontology-Driven Information System Engineering (ODISE), London, UK.

- Jones, D., Oconnor, A., Abgaz, Y., and Lewis, D. (2011). A semantic model for integrated content management, localization and language technology processing. In 2nd Workshop on the Multilingual Semantic Web (MSW2011).

- Abgaz, Y., Javed, M., and Pahl, C. (2010). Empirical analysis of impacts of instance-driven changes in ontologies. In On the Move to Meaningful Internet Systems: OTM 2010 Workshops, Lecture Notes in Computer Science.
- Javed,M., Abgaz, Y., and Pahl, C. (2010). Ontology-based domain modelling for consistent content change management. In International Conference on Ontological and Semantic Engineering (ICOSE).
- Pahl, C., Javed, M., and Abgaz, Y. (2010). Utilizing ontology-based modelling for learning content management. In Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010 (pp. 1274 - 1279). Toronto, Canada: AACE.
- Javed, M., Abgaz, Y., and Pahl, C. (2009). A pattern-based framework of change operators for ontology evolution. In On the Move to Meaningful Internet Systems: OTM 2009 Workshops, volume 5872 of Lecture Notes in Computer Science (pp. 544 - 553).

Glossary of Terms

ABox :	The ABox contains extensional knowledge about the domain of interest, that is, assertions about individuals, usually called membership assertion [Baader et al., 2003]
CMS:	Content management systems are systems that are built to organize, store, retrieve and present content.
Complete Change:	Complete change is a change which is the union of the requested change and the derived changes.
Consistency:	An ABox A is consistent with respect to a TBox T , if there is an interpretation that is a model of both A and T . We simply say that A is consistent if it is consistent with respect to the TBox
Derived Change:	Derived changes are changes that are automatically generated to correctly implement the requested change in a given ontology.
Effect:	Effect is the consequence of applying an action. In this context the action is the implementation of a change operation.
Entity:	Entity refers to the constructs of the ontology, the annotation and the content. An entity refers to concepts, object properties, data properties, instances, content documents, axioms and restrictions.
Impact:	The term impact refers to the effect of change of entities due to the application of a change operation on one or more of the entities in the OCMS
OCMS:	Content management systems that use ontologies to enrich the semantics of the content. OCMSs use the semantics for facilitating information browsing, retrieval and reasoning services.
Requested Change:	A requested change is a change which is captured as an explicit change request.
Satisfiability :	A concept C is satisfiable with respect to T if there exists a model I of T such that C^I is non empty. In this case we say also that I is a model of C [Baader et al., 2003].
Severity:	Severity measures the intensity or the degree of an impact on an OCMS in relation to the problem it causes, the effort and the level of expertise it requires to resolve the impact.
TBox :	The TBox contains intentional knowledge in the form of a terminology and is built through declarations that describe general properties of concepts [Baader et al., 2003].

List of Acronyms and Abbreviations

$\mathcal{AB}$ox :	Assertion Box
CIA:	Change Impact Analysis
CMS :	Content Management System
CNGL :	Centre for Next Generation Localization
CVA :	Close Vocabulary Assumption
DAML :	DARPA Markup Language
DL :	Description Logic
DTD :	Document Type Definition
IRI :	Internationalized Resource Identifier
KAON :	KArlsruhe ONtology and semantic web tool kit
KR :	Knowledge Representation
LOF:	Layered Operator Framework
MILO :	MId-Level Ontology
OCMS :	Ontology-based Content Management System
OIL :	Ontology Interchange Language
OVA :	Open Vocabulary Assumption
OWL :	Web Ontology Language
RACER :	Renamed $\mathcal{AB}$ox and Concept Expression Reasoner
RDF :	Resource Description Framework
SHOE:	Simple HTML Ontology Extension
SUMO :	Suggested Upper Merged Ontology
$\mathcal{TB}$ox :	Terminology Box
URI :	Uniform Resource Identifier
URL :	Uniform Resource Locater
WIS :	Web Information System
XHTML :	eXtensible HyperText Markup Language
XML :	eXtensible Markup Language
XOL :	Xml-based Ontology exchange Language

Chapter 1

Introduction

1.1 Motivation

Every day, new things emerge and others vanish or change. There are plenty of new innovations introducing new disciplines, concepts, objects, devices, services, etc., or altering existing ones to serve a new purpose. Despite the innovations, there are changes that discard existing entities that are not capable of adapting themselves to the changing requirements of the environment. Any human innovation is subject to a continuous evolution and adaptation to the changing requirements of human beings over time.

Human knowledge is subject to change throughout history. Knowledge is growing very fast[1] introducing previously unknown information about entities, consolidating existing ones or abandoning the obsolete ones. The falsification of existing claims, beliefs and theories that are found wrong, unfitting and useless become obsolete and are discarded from the existing knowledge repositories.

At this age of knowledge intensive societies, the use of systems, tools and techniques that facilitate efficient exploitation of the available knowledge for education, business intelligence, research, governance, etc., become ubiquitous on the web [Berners-Lee et al., 2001] [Shadbolt et al., 2006]. Ontologies serve this purpose by representing human knowledge in a formal language which ensures a common understanding among humans and across

[1]http://www.economist.com/node/15557421

machines [Jurisica et al., 1999] [Leenheer & Mens, 2008] [Gross et al., 2009]. The use of ontologies extends this to include very complex artificial intelligence applications that enable machines to understand semantics and perform in a context.

We use ontologies for knowledge representation and semantic annotation to enrich content with information which can be interpreted by machines. Ontologies facilitate the interpretation of the content with a given context and extracting new knowledge from existing ones using semantic reasoning. This is achieved by explicit annotation of concepts in documents with generic and domain-specific ontologies. However, the continuous process of change of content, annotations and ontologies poses a challenge to the efficient exploitation of many of these ontology-based applications [Liang et al., 2006] [Flouris et al., 2008].

Changes occur in the content, the annotation or the ontologies. Content authors may add new sections, edit existing ones or remove unwanted or erroneous parts of their content. In ontologies, changes may occur on the concepts, properties, instances, axioms, etc. Annotations change when either the target content changes or the semantics of the content changes. For example, a personal home page of a professor changes whenever he/she adds new publications, modifies his/her research interest or updates the courses he/she delivers. During the implementation of these changes, the semantics of the content on the page changes accordingly. Another possibility of a change is when the professor changes the source of the semantics (the ontology). If the ontology evolves to include new discoveries, better representation of existing knowledge, etc., the annotation will become vulnerable to change.

In general, the changes are additions of new content, ontologies or annotations, modifications of existing ones or removal of obsolete or erroneous representations. Whenever there is a change of one entity, it may cause many unseen and undesired impacts on other related entities [Gruhn et al., 1995] [Stojanovic, 2004]. The term impact refers to the effect of change of entities due to the application of a change operation on one or more of the entities in a given system [Plessers et al., 2007] [Qin & Atluri, 2009] [Hassan et al., 2010]. It is arduous and time consuming to manually catch these impacts. If we ignore them, they may cause inconsistencies [Konstantinidis et al., 2008], invalidities [Qin & Atluri, 2009] and changes of semantics in the ontology. Thus, before we implement the changes, it is vital to conduct a change impact analysis to understand which entities are affected and how they are affected [Khattak et al., 2010].

1.2 Research Context

An ontology is a specification of a shared conceptualization of a domain [Gruber, 1993]. Ontologies are used to explicitly represent human knowledge using formal languages understandable by humans and machines [Berners-Lee et al., 2001] [Shadbolt et al., 2006]. Ontologies are further used to define concepts in a domain to reach at a common understanding on subjects of interest. Information retrieval, social networks, software development, content management, linked data, artificial intelligence, etc., are some of the domains that use ontologies to semantically enhance and to conceptually structure existing knowledge and systematically infer a new one. Ontologies further serve as conceptual models for organizing concepts in different domains. They are also used to semantically describe entities using domain-specific contexts [Holohan et al., 2006] [Pahl et al., 2007].

In content management systems, ontologies are used to semantically enrich content by explicitly linking the content with the ontology [Oliver et al., 1999] [Noy & Klein, 2004] [Uren et al., 2006]. This semantically rich content is used by humans and computers to better understand and exploit the content. The formal and explicit linkage of fragments of content with entities in an ontology is referred to as semantic annotation. Semantic annotation embeds additional information to a given content. This information can be used to further describe and reason about the content.

Content refers to any information that is published or distributed in a digital form, including text, sound recordings, photographs, images, motion pictures, video and software [Boyce & Pahl, 2007]. We adapted this definition to refer to any digital information that is in a textual format that contains structured or semi-structured documents, web pages, executable programs, software help files, etc. With semantic annotation of content, the use of ontologies is becoming widespread in content management systems [Uren et al., 2006]

[Oren et al., 2006].

Content management systems are systems that are built to organize, store, retrieve and present content. Such systems vary from a simple file-based system to complex database systems. Content management systems further vary in the service they provide. Some of them provide a simple store and search functionality and others go beyond that and provide semantics to the content. They use the semantics for facilitating information browsing, retrieval and reasoning services. We call such systems Ontology-based Content Management Systems (OCMS) to distinguish them from the traditional Content Management Systems (CMS). Thus, OCMS are content management systems that are built using ontologies and semantic annotations to embed meaning and context in the content documents.

In OCMS, the ontologies play a major role in providing semantics to the concepts within a given domain. The concepts in the ontologies aggregate instances which have a similar behaviour. The properties further explain the attributes of the concepts. These attributes can be attributes that describe a concept or its relationship with other concepts. The ontologies further provide a means to specify the nature of the relationship together with restrictions imposed on the relationship.

OCMS play an important role in realizing the accessibility, delivery and use of information over the semantic web. OCMS not only provide content but also provide the semantics associated with the content using a formal representation that can be interpreted by both humans and computers. When we present the content, we encode additional annotations which are useful for understanding the semantics and the properties of the content and their interpretation by systems that consume them. This is achieved by inference using the ontologies and the annotations.

1.3 Challenges and Problem Statements

Since ontologies can be shared and reused across different applications and groups, they are used frequently on the semantic web to ensure the consistent use of concepts within and among different systems [Stojanovic et al., 2002a]. This is achieved using ontologies as a backbone for annotation of the content on the semantic web [Reeve & Han, 2005]. OCMS provide such desired functionality. However, the use of OCMS for pursuing such a purpose is a challenging task. Thus, our research focuses on addressing the following challenges.

The first challenge emanates from the changing nature of the content. Domain content changes frequently introducing changes in the interpretation of the entities. Changes that occur in a domain content may trigger other changes in the content or in the ontology. The changes in the content introduce new concepts, a new way of using already existing concepts or remove existing concepts from the ontology. For example, when new software products, text books, reports, scientific results, etc., appear, we evolve the content to support such requirements.

The second challenge comes from the ontology. Ontology change refers to the change in the specification, conceptualization or representation of the knowledge in the ontology and the implementation of the changes and the management of their effects in dependent ontologies, services, applications, agents, etc. [Flouris et al., 2006]. The overall process of adaptation of ontologies to change patterns and the consistent management of these changes is called ontology evolution [Stojanovic, 2004]. Researchers in the area of ontology evolution have made attempts to make ontology evolution a smooth process. They have suggested several solutions from different perspectives [Klein et al., 2002] [Stojanovic, 2004] [Plessers et al., 2007] [Konstantinidis et al., 2008] [Qin & Atluri, 2009]. The work done in the area of ontology evolution is not yet mature. The analysis of impacts of changes in evolving ontologies does not yet get significant coverage [Khattak et al., 2010]. The first change focuses on bottom-up changes and the second change is top-down change.

The third challenge comes from the annotation. The annotation, which links the content with the ontology, changes frequently leaving the whole system in a continuous evolution [Uren et al., 2006]. Whenever the semantics of the content changes or entities in the ontology change, the annotation needs to adapt to the changes. This change is frequent and needs to be addressed with sufficient depth.

The fourth challenge is the fast growth of semantic web applications. The fast growth of semantic web applications serves to fuel a significant demand for systems that use ontologies as a key tool to manage content-based systems. This fast growth makes the evolution difficult and time consuming. This is due to the huge effort associated with evolving the ontologies and synchronizing the content to keep the service up-to-date.

In general, providing timely, consistent and reliable information to the users of OCMS is crucial. To ensure this, we need to come up with an efficient method which enables us to respond to changes. To this end, changes need to be represented using change operations that are capable of doing a specific task and which can be combined together to perform complex change requests.

Additionally, it should be possible to analyse the impacts of the change operations on other components and dependent systems of the OCMS. OCMS provide semantic information to other systems. Such systems depend on the OCMS for getting the semantics and making decisions based on the semantics they receive. In such situations, the changes in the OCMS propagate to the dependent systems. It is crucial to understand these changes and identify their impacts before the changes are implemented.

Furthermore, whenever we have more than one implementation strategy of the change operation, it becomes important to measure the cost of evolution of each implementation strategy and select the optimal one. An optimal solution uses criteria such as number of changes, impacts of changes, accuracy and adequacy. For systems that are built to run in a real-time environment, identifying the optimal implementation of changes is important. However, such requirements are not yet addressed and call for a solution.

Thus, the main focus of this research is to examine and develop methods, techniques, tools and algorithms to analyse the impact of change operations in ontology-based content management systems to ensure consistent and predictable evolution of the content and the ontologies. In line with this, we investigate how changes are represented, how they affect the integrity and how we choose an optimal implementation of the change when we have more than one implementation strategy to follow.

1.4 Overview of the Research

1.4.1 Research Hypothesis

The explicit representation of changes in ontology-based content management systems and the analysis of impacts of these changes before their implementation could improve and facilitate a consistent, transparent and predictable evolution of ontology-based content management systems in terms of accuracy, adequacy and integrity of the system.

1.4.2 Research Objectives

The general objective of this research is to develop a change impact analysis method for evolving ontology-based content management systems to ensure consistent transparent and predictable evolution to ensure accurate, adequate, reliable, and optimal solution. The specific objectives of the research are:

- to capture and represent requested change operations accurately and adequately.
- to analyse dependency of entities and analyse impacts of individual, composite and domain-specific change operations in an accurate and predictable way.
- to develop methods that evolve ontology-based content management systems in a consistent, accurate and transparent manner.
- to build up methods for analysis and selection of an optimal implementation of change operations in terms of impacts.

To achieve the above objectives, we use three case studies for problem elicitation, requirement analysis and evaluation of the proposed solutions.

1.5 Research Approach

The change impact analysis framework presented in this dissertation follows a bottom-up impact analysis process. It begins with analysing impacts of atomic change operations and moves up to composite change operations. The framework begins with impact analysis by capturing the user's change request and representing them using change operations. Requested changes are processed and additional change operations are generated using different evolution strategies and dependency analysis. In this phase, the framework generates all change operations that are required to implement the requested change. It analyses the impacts of all change operations using the change impact analysis method. Finally, the framework uses the impacts for the selection of an optimal implementation strategy for the requested change operation. Each of the stages uses different selected approaches to address the specific problems at hand.

First, the change impact analysis framework captures the requested change and represents it using change operations. The change operations are organized into different layers using a layered operator framework. We use this framework to define and clearly represent atomic, composite, domain-specific and abstract changes.

The next stage employs an empirical study for the identification and characterization of the impacts of these atomic change operations. It is used to identify impacts of different change operations using different scenarios in various domain-specific ontology-based content management systems. We further identify the characteristics of the impacts. This includes identifying the change operations that cause a specific impact, defining the conditions at which the impacts occur and identifying the entities that are impacted by the change operations. We use a formal notation to represent the impacts and the preconditions for the impacts.

The core change impact analysis process uses dependency analysis to identify the dependent entities that are affected by the change operation. The effects of a change propagate to dependent entities. However, identifying those dependent entities and the type of the dependency needs a detailed study of the characteristics of the dependencies. Thus, we conducted an empirical study to understand the relationship between the dependencies and the impacts. Dependency analysis allows us to understand the dependent entities and enables us to find the nature of the dependency. This further assists us to determine the impact of the change operation on the dependent entities.

The impact analysis begins from the bottom of the layered operator framework by analysing the impacts of individual atomic change operations. Then, it goes up to determine the impacts of two or more change operations as composite and domain-specific change operations. The impacts of the atomic change operations represent the impact of the operation when a single change is implemented separately and individually. However, when the change is part of another composite change, it only tells us little information about the overall impact. To get a full understanding of the impacts of the changes when they are executed as composite and domain-specific change operations, we need to further analyse the impacts using a composite change impact analysis method.

Composite change operations are aggregations of two or more atomic change operations. However, the impacts of composite change operations are not the aggregation of impacts of atomic change operations. When a composite change operation is implemented, the impacts of the composite change may not be the same as the sum of the impacts of its constituent atomic change operations. Composite change impact analysis identifies techniques to analyse the impacts of composite change operations. To analyse these impacts we employ novel techniques such as impact cancellation, impact balancing and impact transformation.

Finally, the selection of optimal implementation of changes using different change operations is conducted using quantitative analysis of severity of impacts, affected statement types, types of change operations and number of change operations. We use experimental observation to determine the optimal strategy for implementing change operations.

This research covers change representation and analysis of impacts of ontology-based content management systems. It focuses on structural and semantic impacts which include impacts on the satisfiability of the $\mathcal{TB}$ox and the consistency of the $\mathcal{AB}$ox statements. The research also focuses on proposing the optimal implementation strategy for execution of changes at hand. For the empirical study, we used domain-specific ontologies and content which is organized using our OCMS architecture. Here, we specifically focus on changes that are requested by the user and excluded implicit detection of changes. Although visualization of effects of changes is beneficial for the ontology engineer or for the user, we focus on the analysis of the impacts and recommendation of alternative solutions. Thus, visualization and presentation of the impacts is beyond the scope of the research. Despite the need for supporting all kinds of content, we restrict the scope to structured and semi-structured content. For the purpose of the experiment, we specifically focus on HTML and XML content.

1.6 Contribution of the Research

This research has the following major contributions.

- The first contribution is a layered OCMS framework. This framework structures the components of the OCMS to ensure transparent, predictable and traceable evolution.

- The second contribution is a change impact analysis framework. The framework follows a novel approach to analyse impacts of change operations. The framework incorporates change representation, impact analysis and change implementation together with integrity analysis and ensures independent evolution of OCMS components.

- The third contribution is a bottom-up approach to analyse the impacts of changes in OCMS. The change impact analysis process begins by analysing the impacts of atomic change operations. Since it is built on top of the atomic change operations, it ensures the maximum flexibility and expandability to introduce additional composite change operations. This approach is a novel approach for analysing impacts of changes in ontologies and ontology-based applications. The analysis includes a consistent evolution of the ontologies, the content and the annotations by keeping the overall integrity of the OCMS.

- The fourth contribution of the research is a better understanding of the preconditions of impacts and explanations why specific impacts occur. The impact analysis process further identifies the preconditions associated to each impact. When the preconditions are satisfied, we determine the reasons why that specific impact occurs and use the information for exploring alternatives to solve the problem.

- Finally, we contribute toward a model which estimates the cost of evolution and which is used to select an optimal evolution strategy using cost of evolution. The change optimization and implementation model provides a quantitative measure of impacts in ontology evolution.

1.7 Outline

The organization of this dissertation begins with introducing the available languages, tools and techniques for delivering semantically rich content. Then, we move to review state-of-the-art research conducted in the area. We discuss our framework which includes change capturing and representation, change impact analysis and optimal change implementation. The last chapter gives conclusions and future directions. The organization of each of the chapters is given below.

Chapter 2 discusses the available semantic web languages, their syntax and semantics. It gives a brief overview of ontology languages, description logic, OWL2 constructs, ontology editors, ontology APIs and reasoners. Content management systems and annotation tools and platforms are discussed in this chapter.

Chapter 3 gives an overview of content-based systems, ontology evolution and semantic annotation in general. It presents a detailed account of related research conducted in the area and identifies the gap which is not yet covered by the state-of-the-art.

Chapter 4 introduces ontology-based content management systems in general and discusses the layered OCMS framework and its individual constructs in particular. The formal representation of the OCMS using graphs is discussed in this chapter. This chapter further introduces a layered operator framework which organizes the changes into atomic, composite, domain-specific and abstract layers.

Chapter 5 presents the change impact analysis framework. This chapter focuses on the first phase of the framework which includes dependency analysis and evolution strategies. Dependencies which are useful for change impact analysis are discussed in detail. Different dependency types and algorithms to identify dependent entities are discussed. This chapter further discusses customized evolution strategies. The strategies and the dependency analysis results are the major inputs for representing requested change operations.

Chapter 6 presents the change impact analysis process. The individual change impact analysis, the composite change impact analysis and the rules that are used to analyse structural and semantic impacts are presented. The novel contribution of this chapter is the analysis of change impacts to identify the impacts of the requested change operations on the structure and the semantics of the OCMS. The analysis further identifies changes that create unsatisfiability of the $\mathcal{TB}$ox and inconsistency of the $\mathcal{AB}$ox statements.

Chapter 7 presents the implementation of the final change operations. This chapter focuses on efficient utilization of the information gained from the change impact analysis and using it to select the optimal evolution strategy to implement the requested change operation. This phase searches a strategy that minimizes the impacts of the change operations and allows the user to compare between different options.

Chapter 8 gives conclusions, recommendations, and discusses the limitations of the research. It highlights future directions of the research.

Appendix A, B and C discuss the empirical study. The empirical study uses case studies from three domains and exploits them to understand what, how, when and why an OCMS evolves. It is also used to evaluate the proposed solution. The case studies enable the reader to gain familiarity with the use cases and understand the solutions presented in this research.

Appendix D and E presents further analysis results and the questionnaire used for the evaluation respectively.

Chapter 2

Background of the Study

2.1 Introduction

One of the challenges of the information age is the availability of too much information called information overload [Edmunds & Morris, 2000] [Eppler & Mengis, 2004]. The sheer volume of information available and our ability to process and use the available information has shown a wide gap. To curb this problem, the semantic web is proposed as an extension of the current web in which information is given well-defined meaning, better accessibility and improved usage [Berners-Lee et al., 2001].

The semantic web represents the technological standard for operating ontologies in modern information systems. It incorporates wide range of languages and tools that are used by ontologies. It is important to provide an overview of the semantic web tools and technologies for a clear discussion of an OCMS.

The semantic web is defined as "A web of actionable information - information derived from data through a semantic theory for interpreting the symbols. The semantic theory provides an account of 'meaning' in which the logical connection of terms establishes interoperability between systems" [Shadbolt et al., 2006, p.1]. The semantic web provides access to data to be shared and reused by humans and agents by attaching metadata with web resources [Bechhofer et al., 2002].

In this chapter we briefly introduce semantic web technologies, languages and tools that are relevant to this research. The introduction highlights relevant languages and technologies that are used throughout this research to develop, implement and test the proposed methods, techniques and algorithms using a prototype. This chapter serves as a review of existing tools and technologies. It is used as an input for systematic selection of tools and technologies to be used throughout this research.

This chapter is organized as follows. Section 2.2 discusses the available semantic web languages and their syntax and semantics. In Section 2.3 we describe the details of OWL (Web Ontology Language) sub languages, profiles and constructs. In Section 2.4 we discuss description logic constructs and Section 2.5 focuses on ontology editors, ontology APIs and reasoners. Section 2.6 discusses the semantic annotation platforms and tools. Finally we give a summary of the chapter in Section 2.7.

2.2 Semantic Web Languages

There are various semantic web languages developed for the realization of the semantic web[1] [Gomez-Perez & Corcho, 2002]. They serve as a standard languages of communication on the semantic web (Figure 2.1[2]). The semantic web uses these languages as a means of delivering content, and more information about the content elements. Many of these languages serve as standard for communicating information between different agents. The widely used and standardized languages which are related to ontologies and the semantic web are discussed below.

2.2.1 XML, XML Schema and DTDs

EXtensible Mark-up Language (XML) was developed and recommended by W3C in 1998[3]. XML was developed to overcome the limitations faced by Hypertext Mark-up Language (HTML). The major limitation was the lack of extensibility of HTML to include user defined features. XML is used to structure texts and exchange data on the web allowing better

[1]http://www.w3.org
[2]http://www.w3.org
[3]http://www.w3.org/TR/1998/REC-xml-19980210

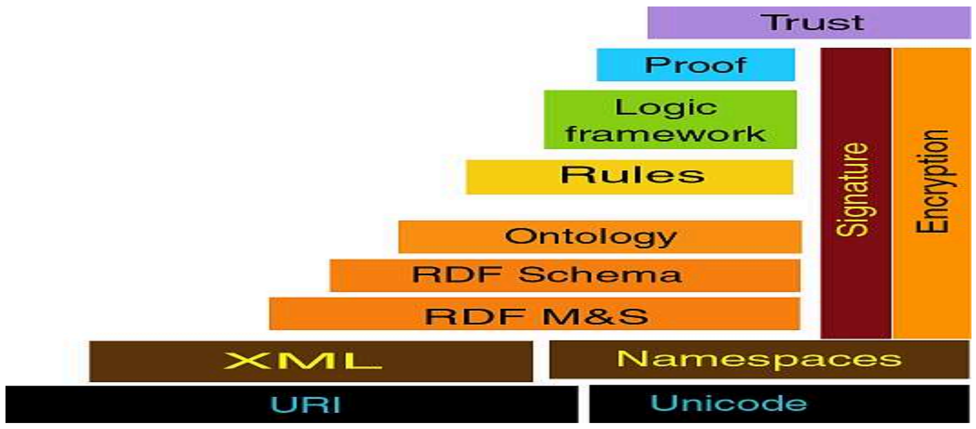


Figure 2.1: The semantic web layers

information exchange across information systems. XML tags are different from HTML tags in that XML tags are user defined and extensible. When XML is used for data exchange between different agents (machines and software), the agents require agreement on the vocabulary and the meaning before they use the data. However, plain XML does not provide such facility. To facilitate the agreement, XML schema and Document Type Definition (DTD) are proposed. XML schema and DTD provide a solution for specifying the structure of XML documents and how they can be used[4]. XML schema and DTD further serve as a mechanism for ensuring the validity of XML documents.

2.2.2 RDF and RDFS

Resource Description Framework (RDF) was developed by the W3C to support the creation of metadata for describing web resources [Gómez-Pérez et al., 2007]. RDF is the widely used standard semantic framework for representing information in the web. RDF describes resources using object types which contain resources, properties and statements. RDF is intended to achieve a simple data model which uses formal semantics that can be proved using inference. It uses XML-based syntax and XML schema. RDF utilizes URI based vocabulary and allows anyone to make a statement about a resource.

A single statement which contains a subject, a predicate and an object can be repre-

[4]http://www.w3.org/XML/Schema

sented by RDF using a statement called triple. Each triple has three elements, the subject indicating the resource we want to describe, a predicate, which is also called a property to specify the relationships the subject has and an object to which the subject relates to using the predicate. All RDF triples can be represented using a graph data model. The graph data model contains nodes and directed edges from one node to another node. An RDF graph is a set or RDF triples which contains subjects and objects as a node and predicates as edges. Figure 2.2 illustrates the representation of RDF triples using RDF graphs.

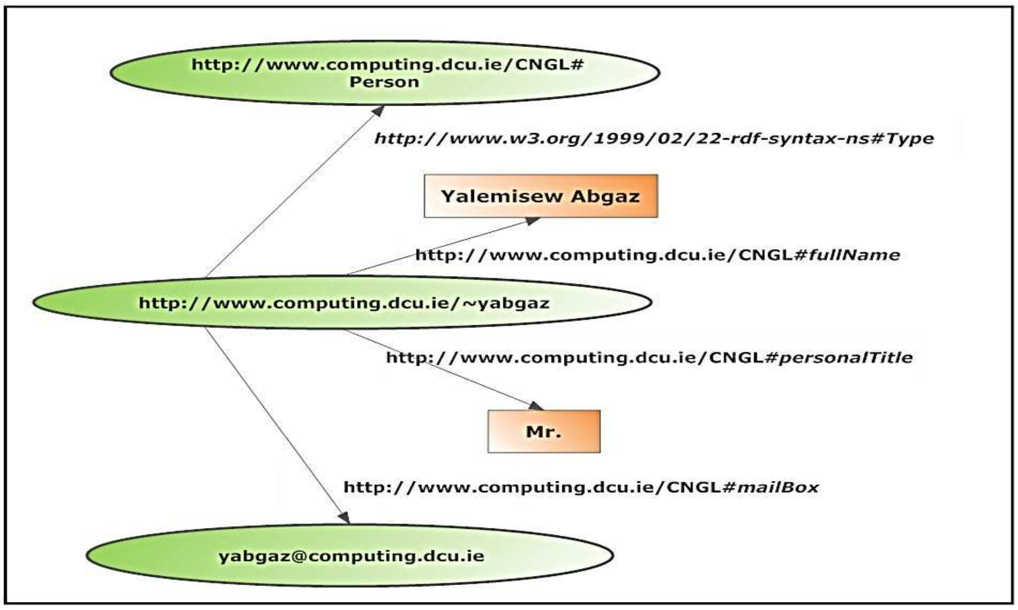

http://www.computing.dcu.ie/CNGL#Person
http://www.w3.org/1999/02/22-rdf-syntax-ns#Type
Yalemisew Abgaz
http://www.computing.dcu.ie/CNGL#fullName
http://www.computing.dcu.ie/~yabgaz
http://www.computing.dcu.ie/CNGL#personalTitle
Mr.
http://www.computing.dcu.ie/CNGL#mailBox
yabgaz@computing.dcu.ie

Figure 2.2: RDF graph representation

The resource Description Framework Schema (RDFS) is a formal description of eligible RDF expressions and a semantic extension of RDF [Gómez-Pérez et al., 2007]. An RDFS provides semantics to describe groups of related resources and the relationships between these resources[5]. The schema is used to determine the characteristics of other resources. The schema provides a list of vocabularies that specify these characteristics using classes and properties.

[5]w3.org/TR/rdf-schema

2.2.3 OIL and DAML+OIL

Ontology Interchange Language (OIL) is also known as Ontology Interface Layer. It is a web-based Knowledge representation language that combines XML syntax, modelling primitives from the frame-based knowledge representation paradigm and the formal semantics and reasoning support of descriptive logics. In OIL, the knowledge contained in the ontology is organized into three parts. OIL combines formal semantics and efficient reasoning support from description logic, rich modelling primitives from frame-based knowledge representation and a standardized syntactic exchange of notations from the web community [Fensel et al., 2001].

DAML+OIL was developed in a collaboration between a joint committee from European Union and the United States of America [Davies et al., 2003] [Mcguinness et al., 2002]. The knowledge representation in DAML+OIL exploits XML and RDF standards and combines formal semantics from description logic, ontological primitives of object oriented and frame-based systems. DAML-ONT is the first version released in 2000 and DAML-OIL is the second version released in 2001. Another version was released by fixing the problems which are related to the specifications in the second version. The DAML+OIL Language is written in XML syntax, unlike the OIL which is written as plain English. The development of DAML+OIL is ceased.

2.3 Web Ontology Language (OWL)

OWL is a web ontology language designed by the W3C web ontology working group for publishing and exchanging of ontologies on the web. It is derived from the DAML+OIL by the standardization efforts of W3C. OWL facilitates the interpretability of web content by providing additional vocabulary and formal semantics [Kruk & McDaniel, 2009]. Unlike the above languages, OWL can be used to represent meanings of terms explicitly and define relationships among the terms [Taye, 2010]. OWL is an ontology language that allows humans to represent semantics of content on the web. It also allows machines to interpret the content. OWL has three sub languages based on the purpose and the available constructs.

2.3.1 OWL sub languages

OWL appears in three different sub languages. OWL Lite is a subset of OWL DL and OWL DL is a subset of OWL full. In general OWL refers to the complete OWL Full language. Each of them is discussed as follows.

OWL Lite is a subset of OWL which is designed to provide easy implementation of the OWL language. It is intended to provide classification of hierarchies that incorporate simple constraints. OWL Lite is aimed at supporting users who want to build tools that use existing reasoners.

OWL DL is another subset of the OWL language that is designed to provide support for existing description logic specification. OWL DL and OWL Full support the same set of OWL language constructs, but OWL DL requires the separation of classes, properties, individuals and restrictions. OWL DL might be chosen over OWL Full due to the availability of powerful reasoners that use the restrictions provided by the users. It has computational features such as completeness and decidability with maximum expressive power within the description logic fragments.

OWL Full is a complete OWL language which allows the relaxation of the constraints of the description logic reasoners. It provides maximum expressivity with full syntactic freedom. But, there is no means of getting full support of reasoning and it is not decidable.

OWL has some drawbacks. Some of its constructs are very complex. To reduce the complexity, OWL has three different sublanguages which deal with this complexity. It is not easy to use and it is not intuitive to non-expert users. The decidability of OWL is achieved by trading-off its expressiveness.

2.3.2 OWL2 Profiles

OWL2 is the recommendation of the W3C since 2009. OWL2 is a successor or OWL1 and has three profiles[6]. Each of the profiles is restricted to a different sublanguage of OWL2. The first profile is the OWL2 RL (Rule Language) which allows rule based reasoning. The

[6]http://www.w3.org/TR/owl2-profiles/

second profile is the OWL2 QL (Query Language) which supports queries against large volumes of instance data that is stored in relational database systems. The third profile is the OWL2 EL which is aimed at applications that use large ontologies and require intensive reasoning capabilities. EL stands for the family of description logic that provides existential and/or universal quantifications.

2.3.3 OWL Syntax

There are different syntaxes used to represent OWL. The first syntax is the RDF/XML syntax and this is the only syntax which is mandatory to be supported by semantic web tools. The functional syntax is designed to provide easier specification and to act as a foundation for the implementation of OWL2 tools and APIS. The Manchester syntax is another variant that is designed to provide easier readability for non-logicians. The Manchester OWL syntax is concise and does not use the description logic symbols [Horridge et al., 2006]. The OWL/XML syntax is an XML syntax for OWL defined by an XML schema. Turtle is a serialization for the RDF-based syntax. Turtle is a triple based notation which extends from N-Triples. It is designed to provide easier and compact textual representation of RDF graph.

Translation between these abstract syntaxes is available. Most existing editors like protege are able to process all the above syntaxes. RDF/XML is a mandatory syntax and every semantic web tool should support the syntax.

2.3.4 OWL Constructs

OWL has different constructs. Some of the constructs that are defined for OWL Lite and OWL DL are discussed below.

2.3.4.1 Entities

- **Owl:Class** represents a group of individuals that share some properties common among them. A group of individuals who joined a university can be referred to as

Student. The top class which is the class of all individuals is usually referred to as *owl:Thing* and the class that does not have any individual is refereed as *owl:Nothing*.

- **Rdf:Property.** is used to specify relationships between individuals or between individuals and data values. OWL distinguishes between **owl:ObjectProperty** which links instances of one class with instances of another class and **owl:dataProperty** which links instances of a class to instances of a data type. For example, *hasFriend* is an object property which links one *Student* with another and *hasAge* is a data property which links a *Student* with an integer data type.

- **Owl:Individual** represents instances of a class. For example, an individual named *Mark* can be described as an instance of the class *PhDStudent*.

- **Owl:Datatype** represents the type of data a given property can take. This includes built-in datatypes such as xsd:double, xsd:long, xsd:string, etc.

2.3.4.2 Boolean Connectives

- **Owl:InteresectionOf** is used to specify the things created by the intersection of named classes and restrictions. For example, *FirstYearPhDStudent* is an intersection of *FirstYearStudent* and *PhDstudent*.

- **Owl:unionOf** is used to specify the things created by the union of named classes and restrictions.

- **Owl:complementOf** is used to specify that one class is a complement of another class.

- **Owl:oneOf** defines a class using a list of individuals belonging to the class. For example, a class of *Influential_person* oneOf {*Obama*, *Blair*, and *Mandela*} which defines an influential person as one of the individuals mentioned in the list.

2.3.4.3 Class Expression Axioms

- **Rdfs:subclassOf** is used to indicate specialization and generalization between classes. For example, rdfs:subclassOf (*Student*, *Person*) indicates *Student* is a specialized class of *Person* and *Person* is a general class of *Student*.

- **Owl:disjointClasses** indicates two classes are different to each other. They do not share a common individual. For example, owl:DisjointClasses (*Male*, *Female*) specifies that individuals of *Male* class cannot be a member of *Female* class.

- **Owl:equvalentClass** is used to indicate two classes are the same and have same instances. A *FirstYearStudent* can be stated to be equivalent class of *FreshManStudent*. If *John* is an instance of *FirstYearStudent*, it can be inferred that he is also a *FreshManStudent*.

2.3.4.4 Property Axioms

- **Rdfs:subPropertyOf** is used to create a hierarchy between properties. It has subDataProperty and subObjectProperty constructs.

- **Owl:EquivalentProperty** is used to specify that two properties are the same and relate to the same set of individuals (domain) to another set of individuals (range).

- **Owl:DisjointProperty** is used to specify that two individuals are not allowed to relate to each other with both properties at the same time.

- **Rdfs:Domain** is used to limit the individuals that are linked as a domain of a property. When a class is specified as a domain of a property, the individuals that are linked to the property must belong to that class. For example, if the domain of an object property *teaches* is a class *Lecturer*, and if *John* is related by *teaches* relationship, then it follows that *John* is a *Lecturer* even if *John* is not explicitly stated an instance of a *Lecturer*. rdfs:domain is a universal restriction because the restriction is imposed on the property.

- **Rdfs:range** is used to limit the individuals that are linked as a range of a property. A property can have a class as a range. When a class is specified as a range of a property, other individuals that are linked to the property must belong to the range class. In the example above, if we set the range of the *teaches* property to be *Course*, and if we link *John* with *CS101* by *teaches* property, the reasoner infers *CS101* as an instance of a *Course*.

- **Owl:InverseOf** is used to specify the inverse property of a property. For example, an object property called *hasFriend* may have an inverse property *isFriendOf*. In this case the domain of a property becomes a range of its inverse and vice versa.

- **Owl:TransitiveProperty** is used to specify that the property is transitive. If a property P is transitive and links two pairs of individuals P(I, J) and P(J, K), then P(I, K) is also an instance of the property P. For example, if *hasAncestor* is stated as a transitive property and if *I* has ancestor *J*, and *J* has ancestor *K*, then *I* has ancestor *K*. Such kinds of relationships are expressed by setting the property transitive.

- **Owl:SymmetricProperty** is used to specify properties that are symmetric. A property is symmetric if a pair of individuals *(I, J)* is an instance of a property, then the pair *(J ,I)* is also instance of that property. If *hasFriend* is defined as symmetric and if *John* has a friend *Mark* as *hasFriend(John, Mark)* then it is also true that *Mark* has a friend called *John* as *hasFriend(Mark,John)*.

- **Owl:FunctionalProperty** is used to specify that a property has a unique value. When a property is set to be functional, then each individual that uses this property will have zero or one value. It is a short hand representation of minimum cardinality 0 and maximum cardinality 1.

- **Owl:InverseFunctionalProperty** Is used to state that the inverse of the property is functional. It is used to state unambiguous properties. If *hasIdNumber* is inverse functional for a *Student* class, then its inverse *isIdNumberOf* becomes functional. This states that a single ID number will not be given for more than one student.

2.3.4.5 Restriction Axioms

- **Owl:AllValuesFrom** is used to state that a property on a particular class has a local range restriction associated with it. When an instance of a given class is used in the property, all the ranges that participate in this relationship should come only from a specific class. For example, *Children* hasFather AllValuesFrom *Male* means if an instance of a child participate in the relationship hasFather(*John*, *Joseph*) then the reasoner infers that *Joseph* is a *Male*. The AllValuesFrom restriction is local to the class involved in the relationship as a domain. The property can be used differently with another class. For example, *Cat* hasFather AllValuesFrom *MaleCat* which restrict only individuals of *MaleCat*. AllValuesFrom does not require a child to have a *Father*, but when it has one, the *Father* should be *Male*.

- **Owl:SomeValuesFrom** is used to state that a particular class may have a restriction on a property that at least one value for that property is of a certain type. For example, *TextBook* has *Author* someValuesFrom *professionalWriter*. This means for all textbooks, they have at least one author who is *professionalWriter*. SomeValuesFrom requires the *TextBook* class to have at least one professional author. However, it allows additional authors who are not professional writers.

- **Owl:MinCardinality** is used to state restriction on a property with respect to a particular class. If a minCardinality of *n* is imposed on a property with respect to a class, then any instance of that class will be related to at least *n* individuals by that property. *Author hasPublication* min 1 *Publication*, means an instance of an author should involve the *hasPublication* relation at least once with an instance of *Publication*. When the reasoner gets an instance of *Author*, it deduces that the instance has at least one *Publication*.

- **Owl:MaxCardinality** is used to state restriction on a property with respect to a particular class. If maxCardinality of *n* is imposed on a property with respect to a class, then any instance of that class will be related to at most *n* individuals by that property.

Author hasPublication maxCardinality 2 *Publication*, means an instance of an *Author* should involve in the hasPublication relation at most two times with an instance of *Publication*.

- **Owl:ExactCardinality** is used to state a restriction on a property with respect to a particular class. If exactCardinality of *n* is imposed on a property with respect to a class, this means that any instance of that class will be related to exactly *n* individuals using the property. *SinglePaperAuthor hasPublication* maxCardinality 1 *ResearchPaper*, specify that all instances of a *SinglePaperAuthor* class participate exactly once in (*hasPublication ResearchPaper*).

2.3.4.6 Class Assertion Axioms

- **Owl:SameAs** is used to specify that two individuals are the same. For example, an individual identified by CS101 is the same as an individual identified by *Introduction_to_Computers*. SameAs(*CS101*, *Introduction_to_Computers*).

- **Owl:DifferentFrom** is used to specify that an individual is different from another individual. This is used to explicitly state the different individuals of a given individual.

- **Owl:AllDifferent** is used to represent that all the individuals involved in the list are mutually distinct and are different from every other individual in the list. For example, we may say AllDifferent(*CS101*, *IS101*, *BU101*). This means *CS101* is different from *IS101* and *IS101* is different from *BU101* and *BU101* is different from *CS101*.

2.4 Description Logic Syntax and Semantics

Description Logic (DL) represents a family of knowledge representation (KR) formalisms that represent the knowledge of an application domain (the “world”) by first defining relevant concepts of the domain (its terminology) and then using these concepts to specify properties of objects and individuals occurring in the domain (the world) [Baader et al., 2003].

DL languages are equipped with formal logic-based semantics and emphasize in reasoning services.

A knowledge base comprises two sets of statements: $\mathcal{TB}$ox and $\mathcal{AB}$ox statements. $\mathcal{TB}$ox (Terminology Box) statements are statements that introduce the terminology (vocabulary) used in the application domain. The $\mathcal{TB}$ox statements focus on concepts (classes in OWL) and roles (properties in OWL).

$\mathcal{AB}$ox statements are statements that contain existential knowledge about the domain of interest. They are assertions about individuals. $\mathcal{AB}$ox statements can be concept assertions (class assertion in OWL). For example, *Person (John)* asserts that an individual identified as *John* is a *Person*. $\mathcal{AB}$ox statements can also be role assertions (property assertion). For example, *hasFriend(John, Joseph)* is a property assertion indicating the individual named *John* has an individual named *Joseph* as a friend.

DL languages allow building complex descriptions of concepts and roles that are represented by other atomic concepts and roles. DL languages are distinguished by their description language and the descriptions they support. In DL, elementary descriptions are atomic concepts and atomic roles. [Baader et al., 2003] discusses the description logic languages as follows. Elementary descriptions are atomic concepts (A and B) and atomic roles (R). Concept descriptions are represented using letters C and D. One DL language is different from the other languages by the allowed constructors in the language. AL (attributive language) is introduced as a minimal language that is of practical interest [Baader et al., 2003] [Schmidt-Schaubß& Smolka, 1991].

AL supports the following Syntax rules:

C,D $\longrightarrow A|$ (atomic concept)

$\top|$ (universal concept)

$\perp|$ (bottom concept)

$\neg A|$ (atomic negation)

$C \sqcap D|$ (intersection)

$\forall R.C|$ (value restriction)

$\exists R.\top$ (limited existential quantification)

AL languages allow negation only on atomic concepts and only the top concept is allowed in the scope of existential quantification over a role. OWL DL corresponds to the SHOIN(D) variant of DL languages.

SHOIN stands for:

S = ALC with transitive Role R+

H = role inclusion axiom

O = nominal (singleton class)

I = inverse role R-

N = number restriction

D = use of data type properties, data values or data properties

Reasoning in DL languages. DL systems focus on reasoning about the domain of knowledge they represent. Reasoning about $\mathcal{TB}$ox statements checks whether a given $\mathcal{TB}$ox statement is satisfiable (meaning not contradictory) with respect to other statements. It also checks whether one $\mathcal{TB}$ox statement is more general that another one (one subsuming the other). Reasoning about $\mathcal{AB}$ox statements checks whether the set of assertion statements are consistent against the $\mathcal{TB}$ox statements. It checks whether the statements have a model. Satisfiability checking in the $\mathcal{TB}$ox statements and consistency checking in the $\mathcal{AB}$ox statements are useful to determine the overall consistency of the knowledge base. Subsumption in turn allows the vocabulary to be organized in a hierarchy. A detailed description of DL language inference can be found in [Baader et al., 2003].

2.5 Ontology Editors and APIs

OWL is a complex language and requires a tool support to create and deploy and evolve ontologies. There are different ontology editors available to create, edit, merge and evolve ontologies. We give a brief review of the available and widely used ontology editors.

2.5.1 Ontology Editors

2.5.1.1 Protege

Protege is an open source ontology editor which is available in two forms of modelling ontology; protege-Frames and protege-OWL. It is based on a Java Application Programming Interface (API) and thus can be incorporated into a number of applications. It is also extensible and is available with a number of supportive plug-ins. It runs on different platforms such as Windows, Linux and UNIX. Protege-OWL is a knowledge model provided with a graphical user interface. Using this GUI, developers can create ontologies in OWL. It is closely linked with Jena3 which is a Java framework to build applications for the semantic web. Jena provides RDF and OWL APIs, parsing and storage in relational databases and query engine for executing queries.

The user interface of protege-OWL allows users to create a new ontology with little effort. One can load ontologies of different format such as in XML, RDF or OWL and ontologies can also be saved in different formats which include OWL, RDF, Latex, Turtle, etc. Users may add annotations to ontologies which could be helpful for later search purposes using an annotation search plug-in.

2.5.1.2 KArlsruhe ONtology and Semantic Web Tool (KAON)

KAON was developed by Information Process Engineering (IPE) group at the research centre for information technologies (FZI), Institute of Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe and the Information Management Group (IMG) at the University of Manchester [Volz et al., 2003].

KAON has an API for programmatic management of OWL-DL, SWRL, and F-logic. It includes an inference engine for answering conjunctive queries. KAON provides a standalone server to access distributed ontologies using Remote Method Invocation (RMI). KAON further provides an interface to access other editors such as protege.

To support ontology evolution, KAON provides an option to set up the evolution parameters. An ontology programmer can decide how to respond to the changes when concepts are removed from the ontology, whether the orphaned concepts must be reconnected to the root concept, to a super concept or must be deleted.

2.5.1.3 NeON and Swoop

The NeOn Toolkit has been developed in the course of the EU-funded NeOn project and is currently maintained and distributed by the NeOn Technologies Foundation[7]. It is an open source editor which supports development of ontologies in OWL/RDF. Neon is developed using the eclipse platform. Neon provides many plug-ins for visual modelling, ontology evaluation, ontology learning, ontology matching and reasoning and annotation and documentation. Neon uses the Pellete2 and HermiT3 reasoners to support inference.

Swoop [Kalyanpur et al., 2011] is a Java based ontology editor designed based on the W3C OWL recommendations. It was developed by the Mindswap group at the University of Maryland. Swoop is based on a web architecture and allows loading of multiple ontologies. However, it is not any more supported by Mindswap group and its development has ceased.

2.5.2 Ontology APIs

2.5.2.1 OWL API and Jena API

OWL API is a Java based API for creating, manipulating and serializing OWL Ontologies. It is an open source software available under the LGPL or Apache licenses. OWL API provides parsers for syntaxes defined in the W3C specification such as RDF/XML, OWL/XML, OWL functional syntax, turtle, KRSS and OBO flat file formats. The original version of OWL API supports the OWL1 specifications and the current OWL API supports all the constructs of OWL2 profiles (OWL2 QL, OWL2 EL and OWL2 RL). The main objective of OWL API is to provide OWL editors and OWL reasoners for people who want to build OWL based applications [Horridge & Bechhofer, 2011].

OWL API is designed to make ontology storage easier in flat files, in relational databases and triple stores. It also provides an OWL Reasoner interface to interact with different rea-

[7]http://neon-toolkit.org/wiki/Main_Page

soners. It provides incremental reasoning support that allows reasoners to listen to ontology changes and process them on the fly or queue them in a buffer for later processing. OWL API provides a wrapper class for CEL, FaCT++, HermiT, RacerPro and Pellet.

The Jena API is a programming toolkit developed using the Java programming language. Jena supports semantic web languages such as RDF, DAML+OIL and OWL. Jena provides an interface to use reasoners.

2.5.3 Ontology Reasoners

This section focuses on introducing some of the available semantic web reasoners that are used to classify ontologies. Reasoners are characterized using different criteria such as, reasoning method, the expressiveness, the time and space complexity, the availability of explanation for inconsistencies and the support of $\mathcal{AB}$ox reasoning [Glimm et al., 2010] [Motik et al., 2007].

An ontology reasoner is a program that infers logical consequences from a set of explicitly asserted facts or axioms and typically provides automated support for reasoning tasks such as classification, debugging and querying [Dentler et al., 2011]. $\mathcal{TB}$ox reasoning corresponds to the reasoning of $\mathcal{TB}$ox statements and $\mathcal{AB}$ox reasoning includes $\mathcal{AB}$ox statements in the reasoning process.

2.5.3.1 HermiT

HermiT was developed at the University of Oxford[8]. HermiT is a description logic reasoning system based on an entirely new architecture which addresses the sources of complexity of reasoning. It uses the Hypertableau calculus, which significantly reduces the number of models which must be considered [Glimm et al., 2010] [Motik et al., 2007]. HermiT can determine whether a given ontology is consistent and identifies subsumption relationships between concepts among other features. HermiT is faster in classifying relatively easy-to-process ontologies and even faster when it is applied to more difficult and large ontologies.

[8]http://www.hermit-reasoner.com/

HermiT is an open source Java library and uses an OWL API as an interface and as a parser for OWL files.

2.5.3.2 FaCT++ (Fast Classification of Terminologies)

FaCT++ was developed by the University of Manchester. It is a new generation of OWL DL reasoner. FaCT++ supports OWL DL and a subset of OWL2. The implementation of FaCT++ uses C++ based on an optimized tableaux algorithm [Tsarkov & Horrocks, 2006].

2.5.3.3 Others

Pellet was developed by Clark & Parsia. Pellet is an open source reasoner and is written using Java. It was the first reasoner that supported all OWL DL (SHOIN (D)) and has been extended to OWL2 (SHOIQ (D)). Pellet supports all OWL2 profiles [Sirin et al., 2007].

In addition to the above reasoners, there are other reasoners such as trOWL (tractable reasoning infrastructure for OWL2) [Thomas et al., 2010], RacerPro [Haarslev et al.,] and CEL (Classifier or EL+ ontologies) [Baader et al., 2006].

2.6 Semantic Annotation Platforms and Tools

Semantic annotation is a process of attaching semantics to a document or part of a document to provide additional information about the existing piece of data. Semantic annotation is different from tagging in that it enriches the content in the document with semantic data that is linked to formal and structured knowledge of a domain. It gets the semantics from a general or domain-specific ontology. Semantic annotation provides information in a formal language which can be automatically evaluated and interpreted using inference tools.

2.6.1 Semantic Annotation Platforms

Currently, there are different kinds of annotations and annotation platforms. However, there is no unified model for semantic annotation [Oren et al., 2006]. Annotation can be manual, semi-automatic or fully automatic. Manual annotation allows users to attach annotations to documents manually. Users determine what semantics to add to the documents at hand. Semi-automatic annotation is facilitated by a tool or a program that determines the context or that attaches the semantic data to the documents. In semi-automatic annotation, there is a manual intervention to modify, approve or reject the proposed annotation. An automatic annotation is based on a program that identifies the necessary semantic information and automatically attaches it to the content. Semantic wikis and semantic blogs are examples that attach semantic information to the documents to improve accessibility, retrieval, exchange and reuse of the documents.

Semantic annotation in our context refers to the annotation triples that are results of a semantic annotation. An annotation triple contains a subject, a predicate and an object. The subject usually refers to the documents, the predicate refers to the attributes of the documents and the object refers to the semantics to which the content is associated. The semantic annotation we refer is ontology-based semantic annotation. Ontology-based semantic annotation relies on ontologies as its main source of semantics. For example, a single document can be annotated using an annotation triple as follows. The document identified by *http:www.cngl.ie/research/paper01.html* has subject ontology evolution which is defined in a CNGL ontology.

```
<http:www.cngl.ie/research/paper01.html>
<CNGL#hasSubject> <CNGL#OntologyEvolution>
```

2.6.2 Annotation Tools

There are different annotation tools available. Some of them are manual and others are semi-automatic. Annotea[9], OntoMat[10] and COHSE[11] are manual annotation tools and GATE is semi-automatic annotation tool. The above mentioned tools do not require modification of the original documents; however, the implementation mechanism is different. Annotea uses XPointer to store the annotations but GATE duplicates the document and attaches

[9] http://www.w3.org/2001/Annotea/
[10] http://projects.semwebcentral.org/projects/ontomat/
[11] http://www.aktors.org/technologies/cohse/

the annotations at the end of the duplicated document. Annotea uses a “fixed” annotation schema, but the other annotation tools use other schema like ontologies.

2.6.2.1 KIM and COHSE

KIM is a platform for semantic annotation of documents, data and knowledge developed by ontotext . The tool is based on open source platforms and comes with ontologies, text mining capabilities, annotation tools and user interfaces. KIM[12] is based on tools such as GATE and OWLIM. KIM uses a number of ontologies designed for general-purpose semantic annotation [Kiryakov et al., 2004]. KIM uses the Sesame RDF repository for ontology and knowledge storage. It uses a light weight ontology called KIMO. COHSE (Conceptual Open Hypermedia SErvice) is developed to provide an architecture for semantic web [Bechhofer et al., 2002].

2.6.2.2 Semantic Wikis

Semantic wikis use formally defined annotations and ontological terms. There are different approaches to annotate content in a semantic wiki. Semantic MediaWiki[13], IkeWiki[14], WikSAR[15], SemperWiki[16] are among the wiki variants.

2.7 Summary

This chapter gives a brief introduction of the state-of-the-art semantic web languages and technologies. In this chapter we cover the history of different semantic web languages. We further discussed the detailed constructs of OWL. OWL2 is the language we use throughout this document. We use the OWL DL variant to construct our ontologies for the purpose of the research and the experiments.

[12]Phttp://www.ontotext.com/kim
[13]http://www.mediawiki.org/wiki/MediaWiki
[14]http://semanticweb.org/wiki/IkeWiki
[15]http://semanticweb.org/wiki/WikSAR
[16]http://semanticweb.org/wiki/SemperWiki

Among the semantic web tools, we use the protege ontology editor for ontology construction and testing purposes. Protege is selected because it is a widely used editor for research purposes and is built upon OWL API. We further use swoop and NeON for the purpose of comparative evaluation. We use OWL API and Java for the development of prototypes and for the implementation of our algorithms. OWL API is selected because it provides methods for dealing with ontology specification and change management. OWL API has active support from the research community and became a widely accepted ontology API. The OWL API is written in the Java programming language. This makes Java preferable for developing our prototype. The use of a reasoner is crucial in determining the consistency of the ontology. Thus, we use the Hermit reasoner for its easier integration with OWL API and its mentioned benefits. For the purpose of storage, we choose RDF/XML and the Sesame triple store. RDF/XML is used for storing the ontologies and Sesame triple store for storing the annotation triples. The W3C make RDF/XML a mandatory format and applications are required to support at least RDF/XML. Furthermore, there are different tools to translate RDF/XML into other formats. The Sesame triple store supports integration with Java and OWL API. It is faster and compliant to existing semantic web technologies.

Chapter 3

Literature Review

3.1 Introduction

The use of ontologies to represent knowledge in different domains is gaining considerable acceptance [Jurisica et al., 1999] [Uren et al., 2006] [Mika, 2007]. Ontologies are used as knowledge representation tools for covering generic knowledge such as space, time, measurement and domain-specific knowledge such as genes, publications, finance, etc. Ontologies serve as a means of exchanging knowledge between humans and computers on the semantic web. Different tools and techniques emerge to support the creation, evolution, maintenance and use of ontologies and ontology-based systems. Despite the growing number of tools, techniques, methods and systems, we face a dearth of support and solution for handling the evolution of the knowledge [De Leenheer & Meersman, 2007].

There is a substantial gap between our requirements and existing solutions that support evolution in ontology-based applications [Stojanovic, 2004] [Noy & Klein, 2004]. When the ontologies or the underlying systems evolve, there is limited support to help the user to understand and evaluate the impacts of the changes on the evolving ontology, other dependent ontologies and systems that use the ontologies. Addressing the problem requires focusing on specific issues and unifying the already available and proven methods and techniques. Though, there are available ontology evolution frameworks, most of them focus on ensuring the consistency of evolving ontologies. They give little attention to analysis of impacts and selection of optimal implementation strategies using severity of impacts. Enhancing the frameworks and introducing new techniques and methods is inevitable to achieve the desired evolution. One of the inevitable research directions proposed by different researchers [Stojanovic & Motik, 2002] [Flouris & Plexousakis, 2005] [De Leenheer & Meersman, 2007] [Qin & Atluri, 2009] [Djedidi & Aufaure, 2010b] focuses on change impact analysis in evolving ontologies and ontology-based applications. In this chapter, we review existing research and highlight their limitations.

This chapter is organized as follows. Section 3.2 gives a brief summary of evolution in ontology-based content management systems. Section 3.3 introduces evolution in other closely related disciplines. In Section 3.4, we review existing ontology evolution approaches in detail. In Section 3.5, we give a review of available tools and support for ontology evolution. Finally, we give a summary of the chapter in Section 3.6.

3.2 Evolution in Ontology-based Content Management Systems

Evolution in OCMS requires efficient tools, and techniques to address the challenges. The ever changing human knowledge, its specification and representation, the availability of large amounts of semantically rich data and the growing number of interdependent applications that consume the data make the evolution process complex, arduous and time consuming. This forces researchers to look for automatic or semi-automatic solutions to efficiently evolve OCMS together with dependent applications.

The need for handling changes in OCMS arises from different perspectives. The first one is from an application perspective. The ever growing volume and diversity of content, ontology and annotation demands a shift to a scientifically proven method that can overcome the current problems faced by the manual system [Liang et al., 2006].

The second comes from the perspective of systems and users. In real-time environments, up-to-date and accurate information is required. Many users and systems are dependent on the semantic data they gather, process and use from the semantic web. The quality and the accuracy of their services depend on how much up-to-date information they are using as an input. Up-to-date information can only be provided if changes in such domains are handled, all the effects of the changes are identified and the integrity of the whole system is assured. Such systems rely on the availability of automatic or semi-automatic tool support to respond in a timely manner to the growing number of application domains and semantic data which is in continuous state of change [Klein et al., 2002] [Qin & Atluri, 2009] [Afsharchi & Far, 2006].

The third perspective comes from the nature of the content. Content refers to a broader collection of human knowledge. In some disciplines, the content evolves every single day demanding addition of new and previously unknown content, modification of existing one or deletion of old content. Systems that deal with content of this type are subject to a continuous change which may even lead to the evolution of the ontologies they use. Such dynamic and evolving disciplines, like computer science, become vulnerable to continuous change that demands automated tool support [Stojanovic et al., 2002b] [Leenheer & Mens, 2008].

In addition to the above perspectives, the growing volume of content, applications, domain-specific ontologies and the complexity of changes, the knowledge and time required to identify, understand and handle these changes manually is beyond the comprehension of human agents. Every minute, 571 new websites, 347 new blog posts, 48 hours of new video on YouTube, 204 million emails, 100,000 tweets, 2 million search queries and many other content is produced on the web[1]. Wikipedia alone contains around 4 millions of content pages and around 28 million individual pages. The average edit per page is 19.75[2]. All these new content pages and changes require a solution to systematically handle additions of new content and deletion or modification of existing ones.

It is possible to collect changes over time and apply those changes together on a daily or weekly basis. For content-based systems that do not change frequently, this solution seems appealing. However, the current problem of such systems goes beyond that. First, current ontology-based applications are required to provide semantically rich information to make a real-time decision. Second, the volume of the changes the ontology engineer has to

[1]http://mashable.com/2012/06/22/data-created-every-minute/
[2]http://en.wikipedia.org/wiki/Special:Statistics

deal with is large. In fairly complex environments, such as Wikipedia, there are thousands of changes, hundred thousands of cascaded changes and millions of artefacts, dependent entities and interrelated components that will be affected. In such a situation, managing changes manually becomes complex, error prone and beyond the comprehension of a single individual. In relation to this, systems that are required to deliver robust information do not tolerate these errors. The structural and semantic interdependence that exists between components of such systems requires an automated tool to prevent and resolve issues related to the integrity of the system. Finally, the availability of tools and techniques can be seen as an opportunity to build better automated systems for OCMS.

Evolution in OCMS mainly focuses on the evolution of the ontology, the content and the annotation. Ontology evolution, content evolution and annotation evolution are discussed in detail in the following section.

3.2.1 Ontology Evolution

Ontology evolution refers to the change in the specification, conceptualization or representation of knowledge in the ontology and the implementation of the changes and the management of their effects in dependent ontologies, services, applications, agents or other elements. Ontology evolution is also defined as “the timely adaptation of an ontology and consistent propagation of changes to dependent artefacts” [Stojanovic et al., 2002a]. The overall process of adaptation of ontologies to changed patterns and the consistent management of these changes is referred to as ontology evolution. A closely related task is ontology versioning. Thus, it is important to distinguish between ontology evolution and ontology versioning. Following the definition given by [Roddick, 1995], ontology evolution is different from ontology versioning in that ontology evolution is the process of changing the ontology without affecting the dependent entities whereas ontology versioning means changing the ontology to a new version but provides access to both the old and the new versions.

Ontology evolution is a continuous process [Noy & Klein, 2004]. Whenever there is a change in the domain, its conceptualization or specification, the ontology needs to be changed. Ontologies built to give support for specific content within a domain change as the content and the embedded ontology instances change [Benjamins et al., 2002]. When new concepts are added, existing ones are deleted or modified in the content, the respective ontology needs to be updated. Implementing the changes requires understanding them correctly and representing them accurately using ontology change operations. However, this solves a few of the associated problems. Changes can trigger further cascaded changes and affect one or more of interrelated ontologies. The effects of the change may propagate back to the domain instances in the content, leaving the process in a vicious circle. An ontology engineer who detects a change of an instance in a content document and tries to maintain the ontology accordingly may end up with many unseen changes. Thus, we require a better understanding of the ontology change management process [Djedidi & Aufaure, 2010b].

Researchers in the area of ontology evolution have made attempts to make ontology evolution a well-organized process. In [Stojanovic, 2004] the authors proposed a six phase ontology evolution process. Ontology evolution as a reconfiguration problem is suggested in [Stojanovic et al., 2003]. In [Plessers et al., 2007] change detection approach is proposed as a solution for efficient ontology evolution. In [Qin & Atluri, 2009] the authors approach the problem from the view point of validity of instances at the time of evolution. These attempts are discussed in detail in Section 3.4.

Ontology evolution further focuses on the impacts of the proposed changes on related dependent entities in the ontology and dependent systems that use the ontology. The impacts are not restricted to the structure of the ontology. But, they include the semantics and deal with the rationale behind the changes [Djedidi & Aufaure, 2010b].

3.2.2 Content Evolution

Content change refers to a change of the available content in a content-based system. The change in the content can introduce new concepts, for example, when new software products, text books, reports, scientific results, etc., appear, or when a new way of using already existing concepts is introduced.

Evolution in a discipline invokes a change in the content used in that discipline. The evolution process may cause the previous content to be modified or fully discarded. Content on the web evolves over time in an unpredictable manner due to its decentralized administration. Collaborative content management serves as an engine for managing the continuous evolution of content. Users in a collaborative environment create new content and make it available for others for editing and improvement. Such content passes through different evolutionary stages before it becomes stable [Krötzsch et al., 2007]. In disciplines that evolve frequently, the content evolves frequently. Research on a typical collaborative content platform, Wikipedia, indicates that there are a continuous and a large number of revisions (addition of new ones and deletion or modification of existing ones) of the content [Curino et al., 2008].

There are two types of content changes, changes that cause the domain ontology to change and changes that cause only a specific content to change. To elaborate on these distinctions, let us look at the following examples. A change in a new version of a help file that includes video and audio formats requires a change in the domain to incorporate such help file formats (in previous versions, help files come in text format). An introduction of a new software component further requires the taxonomy of the software to be updated. Such changes are changes that trigger a change in the domain ontology. The deletion of a component from a specific help file causes a change of all annotations that are related to that topic. Not all content changes cause ontology change. If a step in one help file is changed to a procedure, it will only cause a change in the annotation of the document. It does not cause or trigger a change in the domain or in the ontology.

3.2.3 Annotation Evolution

Annotations are frequently changing entities in OCMS. A large number of triples are added, modified or deleted in this layer. Annotations are highly dependent on both the content and the ontology [Gross et al., 2009]. Any change in the annotated content or in the ontology that is used for the annotation affects the annotation triples that carry all the semantics related to the content. Changes made on the triples may cause other changes to the related annotations. In such situations, the changes in the annotation require proper analysis and evolution before they are implemented in the system.

The main reason for propagation of changes is that the annotations represent semantic and structural dependency between the entities involved in the annotation. For example, if we annotate a certain home page of a professor with a domain ontology that explains his discipline, we are creating a semantic link between the concepts in the home page and the concepts in the ontology. The concepts in the ontology are interconnected and get their semantics based on the interpretation of those edges that connect them. Thus whenever there is a change in the home page of the professor, the change propagates to all dependent and related entities in the system. These changes affect the interpretation of the content and all dependent systems that use this interpretation.

3.3 Evolution Approaches in Related Domains

Ontology evolution borrows different techniques from different disciplines. In this section we review and compare literature from closely related disciplines such as schema evolution and software evolution.

3.3.1 Schema Evolution

Research in database schema evolution has shown the extent of the problem and the importance of schema evolution [Roddick, 1995]. Typical schemas include relational database schema, conceptual ER or UML models, ontologies, XML schema, software interfaces and work flow specifications [Noy & Klein, 2004]. [Hartung et al., 2011, p.1] defines schema evolution as "the ability to change a deployed schema, i.e. metadata structures formally describing complex artefacts such as databases, messages, application programs or workflows". Schema evolution has received a great emphasis for a long time and has a wider support from both industrial and commercial systems. The rationale behind conducting schema evolution is to deal with:

- new changes which are caused by a change in the requirement of the user.

- deficiencies in the current schema or the old model may have errors.
- new insights, new ideas, etc.
- migration to a new platform, a new technology, etc.

Schema evolution is essential because the changes are frequent, time consuming and error prone [Curino et al., 2008]. The authors identified the major problems faced by database administrators at the time of evolution. Some of the challenges are attributed to a lack of software support for predicting and evaluating the effects of the proposed schema change, lack of analysis methods and tools for understanding change propagation to dependent elements and applications to address database schema evolution. In schema evolution in general and in database schema evolution in particular, the following solutions are proposed [Bounif & Pottinger, 2006] [Curino et al., 2008].

- Concise change operation language to express schema change.
- Tools that can determine the effects of the requested changes.
- The optimization of the changes to ensure the optimal implementation.
- Automatic implementation and propagation of changes.
- Full documentation of implemented changes to ensure proper and accurate reversibility.

Schema evolution and ontology evolution share some common theoretical foundations. The similarities and the differences of the two are discussed in [Noy & Klein, 2004]. One of the challenging aspects of schema evolution is the threat it poses to the systems that make use of the schema. Some of the threats are unexpected and have dramatic impacts on the dependent elements and on the integrity of the system (within and among the layers). They require intensive human involvement to understand the impacts and resolve integrity violations. These threats are also observed in ontology schema evolution and remain the major challenges faced by ontology engineers and content managers.

[Curino et al., 2008] identified the intensity of the change in Wikipedia's database schema, which is the best-known example of a large family of web information systems (WIS). They identified 170+ documented schema versions over 4.5 years and over 700GB of data and version of 88,397+ revisions in MediaWiki in 2007. They reported the growing frequency of change as: "There is strong pressure toward change (from 39% to 500% more intense than the traditional setting)". To respond to the problem they built a simple software tool to automate the analysis process. Their analysis suggests the need for developing better methods and tools to support schema evolution.

Research conducted by [Hartung et al., 2011] on recent advances in schema and ontology evolution points out the importance of effectively supporting schema evolution to ensure the correct and efficient propagation of changes to instance data, dependent schema and dependent systems. They identified the major requirements of effective schema and ontology evolution. The support for expressive and detailed change operations, the simplicity of change specification, the transparency of the evolution process, automatic generation of evolution mapping and the predictability of effects of changes on instances to maintain data integrity and avoid data loss are among the requirements.

According to the [Hartung et al., 2011] criteria, powerful schema evolution support needs to incorporate the following guidelines. The first one is ***completeness***. Completeness ensures the complete support for schema changes and the correct and efficient propagation of the change to dependent elements and dependent systems. The second is ***minimal user intervention*** which ensures the minimal involvement and automatic evolution of dependent elements and subsystems. The third criterion is ***transparency*** which ensures the minimal or no degradation of performance of the system. It should ensure the availability of support for backward compatibility, versioning or views.

[Hartung et al., 2011] focuses on XML schema evolution and identified systems that provide XML schema evolution support such as Oracle, Microsoft SQL Server, IBM DB2. Some native XML Databases also support XML schema evolution. These commercial software companies are aware of the importance of schema evolution and provide a means to support the evolution process. Ontology evolution is another focus area of the study.

Despite the differences between ontologies, databases and XML, the schema evolution requirements also apply for ontology evolution.

Justifications used in schema evolution highlight the need for ontology evolution. Schema evolution is used to keep the integrity of the data with the schema. Likewise, we evolve the ontology to provide integrity of the instances with the ontology. Compared to schema evolution, the following challenges are identified in ontology evolution [Noy & Klein, 2004].

- When an ontology evolves, it affects the data (instances that are linked to it) and the semantics associated with the ontology. This is because ontologies themselves are treated as a data.
- Ontologies incorporate semantics in their definitions, whenever the semantics they incorporate changes, we need to change the ontologies. In many domains that are relatively new or that deal with new knowledge and information, the change in the semantics is continuous. That calls for a dynamic and continuous evolution of the ontologies to respond to the changing semantics in such domains.
- Current systems that provide semantic information require the annotation of the content using the selected ontology. But, it does not mean that once we get access to the annotated content we no more need to access the ontology. The ontologies are accessed by such systems for the purpose of reasoning and extraction of implicit knowledge.
- Due to the decentralization of the ontologies and the users of the ontologies, it is difficult to know and maintain who makes changes and who needs to be notified about the changes. It is also difficult to update them synchronously.
- Due to the richer semantics they contain, a single change in a single element of an ontology, say on a concept, triggers more changes due to the semantic relationships like disjointness, intersection, transitivity, etc. Thus, managing the changes becomes complex and includes several cascaded change operations.

- Since instances and concepts are not distinctly separate, it is difficult to provide a change management which treats the instances separately from the concepts. This makes the evolution of ontologies difficult.

3.3.2 Software Evolution

Software evolution and ontology evolution share common grounds. Software evolution focuses on evolving the software without invalidating running systems and existing data. In ontology evolution we focus on a similar problem. In software evolution, determining the impacts of the changes on dependent modules, classes and data is the main concern. In ontology evolution, identifying inconsistencies, invalidities and impacts of changes on entities and dependent systems is the main focus.

Software evolution integrates a multidimensional aspect of a software life cycle from the inception phase to maintenance. The dimension includes system properties (what), objects of change (where), temporal properties (when) and change support (how) [Mens et al., 2002] [Buckley et al., 2005]. Software evolution needs a systematic and exhaustive description of the change and the changing artefacts [Mens & Klein, 2012]. Like schema evolution, the process of handling evolution in software is time-consuming and error-prone. The main focus of software evolution is to identify the changing artefact in the software and to identify the artefacts that are affected by the change. Researchers such as [Lehman et al., 1997] [Sherriff & Williams, 2008] [Ahmad et al., 2009] focus on classifying the different components, analysing the dependency between the components and the propagation of the changes to other dependent software artefacts. [Ahmad et al., 2009] uses sets to represent relationships between components. [Sherriff & Williams, 2008] uses association clusters from change records to analyse impacts of changes. These association clusters of files indicate how the files are executed, tested and changed together.

Software systems that embed a software application in the real world, known as E-type systems, evolve frequently. For such systems, researchers [Lehman et al., 1997] identified different laws of software evolution. The first law is about ***continuing change***. E-type systems must continually adapt themselves to changes; otherwise they become progressively less satisfactory. His second law further reinforces the need for software evolution, which is ***increasing complexity***. As a program evolves, its complexity increases unless work is done to maintain or reduce it. These two laws call for a solution to handle the changes and to avoid the associated complexities.

Software change denotes a set of source files that are modified together. The reason for the change may be removal of a defect or introduction of a new feature that reflects the user's requirement. The changes can be logical or structural changes which may affect other dependent components of the software [Wu et al., 2007]. According to the change data the authors analysed on open source products, within 8 years (1997/08/11 to 2005/09/09) they identified 40,034 logical changes from the CVS repository of GCC. The largest logical change obtained from the NetBSD system is 86,280 logical changes from 1993/03/20 to 2005/08/17. Structural changes are also presented and 19,913 changes are identified from the Koffice system from 1999/01/01 to 2004/09/15.

These figures indicate that there is a high frequency of change. It is evident that managing changes and determining impacts of changes is becoming a time consuming and complex task. This implies the need for software tools that deal with the evolution process of software products.

Other research in software evolution focuses on change impact analysis in software systems using empirical analysis [Arnold, 1996] [Lee et al., 2000]. Software change or software evolution has an impact on dependent systems. These impacts were analysed using different techniques such as PathImpact, CoverageImpact and other methods [Bohner, 2002] [Orso et al., 2004] [Breech et al., 2005] [Sherriff & Williams, 2008]. The main aim of software change impact analysis is to find out which dependent components of given software are affected by a change and to take action before the new version of the software is released. Reducing the time and effort of tracking and correcting the erroneous modules is one reason for conducting impact analysis prior to the implementation of the change. As there are more changes and versions in a software product, there are more impacts of these changes on the dependent software components and this needs an automated solution to reduce the impacts.

The author in [Bohner, 2002] conducts a change impact analysis on commercial-Off-The-Shelf software. He identified different reasons for software change and classifies software impacts as direct or indirect, and structural or semantic impacts. The impact analysis method uses graphs to represent dependencies between software components. He uses graphs to analyse structural and semantic impacts of changes on dependent systems. He further conducted structural analysis and semantic analysis using reachability graphs by implementing transitive closure algorithms. The work focuses on the syntactic relationship between software modules whereas we focus on structural and semantic changes with detailed semantics.

In [Hassan et al., 2010], the authors present a knowledge base system for change impact analysis on software architecture. They propose an architectural software component model (ASCM) on which they defined change propagation process. They use graphs to represent a software architecture description represented by ASCM. The graph is used to capture architecture elements and their relationships. The authors conduct an impact analysis using rules that define change propagation. The change propagation process uses a knowledge-base system which stimulates the impact on the software architecture and on the actual code when the associated rules are fired. Their work is similar to ours but with a significant difference in the domain and in the impact determination approach.

To deal with software change and to understand and manage the effects of the changes, different researchers conducted software change impact analysis. [Ren et al., 2004] develop a tool for change impact analysis for Java programs. The authors identify changes by comparing two versions of a program and represent the changes as atomic changes. Using the atomic changes, they analyse the affected elements that changed their behaviour due to the atomic changes and they explain the causes of the effects. The authors use a call graph that represents methods using nodes and edges between nodes to reflect calling relationship between methods. Their approach starts with identifying affected tests (targets), and moves to identify affecting tests (causes). The method computes syntactic dependencies to determine the causes of the changes. Syntactic dependencies and semantic dependencies are independent of each other and are treated differently. Using syntactic and semantic dependencies, they identify the impacts of the changes on the edited version. The impact of a single change when executed alone has a different impact than when it is executed as part of a composite change. However, it is not possible to apply this approach unless we have access to the original and the edited software.

3.4 Evolution Approaches in Ontology-based Applications

Attempts are made to enhance ontology evolution by adopting well established techniques from other disciplines such as database schema evolution and software evolution. However, the techniques borrowed from such disciplines do not fully address the problems of evolving ontologies and ontology-based applications. Ontology evolution and versioning in general and change detection, change representation, change propagation, semantics of change, evolution in distributed ontologies and change impact analysis are among the problems that require further investigation [Stojanovic, 2004] [Djedidi & Aufaure, 2010b].

3.4.1 General Ontology Evolution Approaches

The requirements and the characteristics of ontology evolution have been discussed in different papers [Bennett & Rajlich, 2000] [Stojanovic & Motik, 2002] [Noy & Klein, 2004] [Stojanovic, 2004] [Flouris & Plexousakis, 2005] [Noy et al., 2006] [Lee et al., 2007]. In all these investigations ontology evolution is treated as a complex and non-trivial problem. An ontology evolution begins with capturing a change request and ends with implementing the requested change without invalidating the ontology and dependent systems. Ontology evolution involves several intermediate steps such as change representation, semantics of change, change propagation, change validation and finally, change implementation.

Ontology evolution also distinguishes between different levels of change operations. [Stojanovic, 2004], for example, classifies changes as atomic, composite and complex changes and provides support for the first two categories. Other research [Klein, 2004] distinguishes between basic and complex change operations. Basic change operations represent simple and atomic changes that modify only one specific feature of the ontology, and composite changes represent complex changes that are composed of atomic changes grouped in a certain logical order. Our ontology evolution approach considers four levels of change operations, atomic, composite, domain-specific and abstract. We mainly focus on addition and deletion change operations. Atomic or elementary change operations are finite based on the available constructs of the ontology language. However, composite (complex) and domain-specific changes are infinite as there is no limit on their combination.

The ontology evolution process involves capturing change requests, detecting changes and version logging, change representation, semantics of change, change implementation, change propagation, change validation, and other tasks [Stojanovic, 2004] [Zablith, 2008] [Flouris et al., 2006] [Konstantinidis et al., 2008] [Zablith et al., 2008] [Qin & Atluri, 2009]. Ontology versioning [Klein & Fensel, 2001] [Klein et al., 2002], change impact analysis and resolution, detection of patterns from change logs and others areas are under investigation.

Six phase ontology evolution approach is an ontology evolution approach proposed in the literature and gained wider acceptance as a global evolution process for KAON Ontology [Stojanovic, 2004]. The proposed ontology evolution methodology includes six phases targeted for business-oriented ontology management. The six phases are discussed as follows.

Change capturing. This phase focuses on the process of capturing ontology changes by explicit request from users or implicit change detection and discovery methods. The change detection and discovery method employs changes that are captured either by a data-driven method or by a usage-driven method which analyses the behaviour of the ontology usage patterns [Stojanovic et al., 2003] [Maedche et al., 2003] [Stojanovic, 2004].

Change representation. This phase focuses on the representation of the change operations based on the KAON language. Elementary changes and composite changes are used to represent the change operations. However, the proposed change representation method does not cover domain-specific change operations.

Semantics of change. This phase deals with evaluation and resolving effects of changes to ensure consistency of the whole ontology [Stojanovic, 2004]. This phase enforces consistency rules as invariants that must be satisfied, soft constraints which can be violated for a period of time and user-defined constraints which are defined by the user to accommodate his/her requirements. This phase makes sure that these constraints are satisfied without introducing any inconsistency. [Stojanovic, 2004] [Qin & Atluri, 2009] have identified structural and semantic inconsistencies. Structural inconsistencies are those statements in the ontology that violate the structural constraints defined in the ontology model. Semantic inconsistencies are those statements that alter the meaning of the ontology entities. However, the available methods do not fully deal with semantic inconsistencies. This is because, handling inconsistencies is dependent on specific semantic information that is not explicitly expressed in a standard ontology model [Stojanovic, 2004] [Djedidi & Aufaure, 2010a] [Djedidi & Aufaure, 2010b]. Here, our work also focuses on addressing semantic inconsistencies and semantic impacts by analysing individual change operations and utilizing rules for semantic inconsistencies. We further identify semantic impacts from a combination of more than one atomic change operation.

To address the inconsistency problem, we need to identify the inconsistent entity, determine possible alternative solutions and choose one, and proceed to its implementation. A posterior verification approach for consistency checking verifies the consistency of the ontology after every change is implemented. A priori approach checks the potential violations of preconditions associated with each change operation before the changes are applied. A priori verification approach is cheap compared to a posteriori approach in that posteriori verification is applied to the whole ontology and the resolution needs roll back mechanisms [Flouris et al., 2006]. Furthermore, it is difficult to explain the change impacts and pinpoint the inconsistency associated. In KAON, this phase is implemented as a priori verification based on predefined preconditions [Stojanovic, 2004]. We also follow an apriori approach for both semantic and structural inconsistency and impact identification.

Semantics of change also deals with procedural and declarative inconsistency resolution approaches. The procedural approach is based on the consideration of constraints of the consistency model and the associated rules to satisfy the constraints. This approach considers different evolution strategies to produce additional change operations. Evolution strategies play a major role in allowing the user to flexibly handle changes using a different set of change operations in response to the inconsistency introduced. After all the changes are generated, they are implemented in the ontology. The declarative approach follows a formal change request in the form of positive changes and negative changes. The positive changes are implemented directly in the ontology and when there are inconsistencies the resolution strategy is selected based on the two sets of requested changes (positive changes and the negative changes). Finally, all the possible consistent states of the ontology are ranked based on the ontology engineer's requirements.

Change propagation. Change propagates to dependent artefacts, ontologies and systems that exploit the ontologies. Thus, change propagation deals with propagating the ontology change to dependent artefacts.

Change implementation. The change implementation phase concentrates on the physical implementation of the requested and derived changes. This phase includes logging changes, undo and rollback services.

Change validation. This phase is the final phase which is responsible for the final validation of the applied changes and the acceptance and approval of the changes by the users.

This approach is widely used by ontology engineers. Its main focus is the semantics of change phase. We also focus on this phase to find out structural and semantic impacts and to select optimal resolution strategies using different parameters.

Change detection approach follows two widely used methods of change discovery. The first approach is data-driven change discovery, which relies on the changes that are observed in the corpus data. This approach uses taxonomic analysis, text extraction, relationship mining, etc., to detect changes [Cimiano & Völker, 2005] [Bloehdorn et al., 2006] [Enkhsaikhan et al., 2007]. The second approach is user-driven change discovery, where the users submit change requests based on their common understanding of an evolving domain [Stojanovic et al., 2002a].

BOEMIE. Bootstrapping Ontology Evolution with Multimedia Information Extraction (BOEMIE) is another ontology evolution approach proposed by [Castano et al., 2006]

[Petasis et al., 2009]. This approach aims at automating the process of knowledge acquisition for multimedia content. BOEMIE uses ontology population (adding new instances) and enrichment patterns (adding new concepts, relations and axioms). Once the ontology is populated with new changes, the consistency of the ontology is checked to eliminate contradicting and redundant information. Finally, BOEMIE produces a new version of the ontology that reflects the updates and the newly acquired knowledge and the associated change log [Castano et al., 2006]. This approach does not explicitly support change impact analysis.

Ontology evolution in a distributed environment is another approach proposed and used for distributed ontologies and ontology-based systems [Klein, 2004]. In this approach a global framework is used to manage requested changes and derived changes. Distributed change management systems incorporate additional characteristics that are either different or not available in other approaches. These characteristics are: first, the nature of propagation of a change depends on whether the requested change modifies the specification or conceptualization of the ontology and, second, the definition of consistency and consistency maintenance does not depend any more on one specific feature to preserve.

Ontology evolution as reconfiguration-design problem solving is another approach proposed by [Stojanovic et al., 2003]. The ontology evolution problem is reduced to a graph search where the nodes are evolving ontologies and the edges represent changes that transform the source node into the target node. The approach allows the user to submit complex requests with positive changes and negative changes and provide all the possible ways to resolve the request. The approach uses a consistency model and implements change resolution using an evolution graph that generates multiple options for implementing the change. The selection of the best option is guided by heuristic information.

Belief change principles for ontology evolution. A different approach for ontology evolution based on belief change principles is proposed by [Flouris & Plexousakis, 2005] [Flouris et al., 2006]. The authors argue that the existing ontology evolution approaches are unable to handle change representation and the semantics of change phases. They criticize current work on ontology evolution as a process specializing in helping users to perform changes manually rather than performing the changes automatically. They propose a method to handle ontology evolution without human intervention. Belief change provides necessary formalization for change representation and deals with automatic adaptation of a knowledge base to new knowledge. The belief change theory they proposed is based on the AGM theory initiated by three authors (Alchourron, Gardenfors and Makinson,1985). The focus of belief change is on determining the most rational ways of dealing with changes and on the development of algorithms that automatically update knowledge bases. This approach has a deficiency in representing addition and deletion of concepts, roles and individuals. To resolve this problem the authors suggest a proper selection between Open Vocabulary Assumption (OVA) and Close Vocabulary Assumption (CVA) and a consistent use of the selected vocabulary.

Change detection approach using version logs is another approach presented to address the problem of ontology evolution [Plessers et al., 2007]. A change detection is proposed for OWL DL ontologies. It exploits change logs to detect changes that are not explicitly requested and automatically generates an overview of changes that have occurred based on a set of change definitions. The authors proposed the Change Definition Language (CDL) which is used to represent and query a version log [Plessers et al., 2007].

This approach distinguishes between two kinds of evolution: evolution-on-request and evolution-in-response. Evolution-on-request focuses on modifying the ontology by forwarding a change request by the ontology engineers and evolution-in-response focuses on providing information about depending artefacts changed during the evolution-on-request phase. Evolution-on-request has five phases: change request, consistency maintenance, change detection phase, change recovery phase and change implementation phase.

Evolution-in-response takes into account the changes applied by an ontology engineer and evaluates them to approve or reject the changes. The changes are applied and propagated once they are approved, otherwise rejected. It has three different phases: change detection, cost of evolution and update approval.

Ontology Robustness is an other approach which is suggested to reduce the number of change in ontologies by designing a robust ontology [Ceravolo et al., 2008]. This approach is based on the distinction among stable components and contingent components of the ontology. Ontology robustness in evolution is explained as the minimization of the number of instances to be migrated in the new version of the ontology. The aim of the work is to reduce invalid assertions. This approach can be viewed as an alternative approach to ontology evolution.

Formal RDF/S ontology evolution. This approach presents a formal approach for RDF/s ontology evolution [Konstantinidis et al., 2008]. The work aims at providing an algorithm to determine the effects and side effects of a requested elementary or complex change operations. It focuses on change requests and tries to resolve the evolution problem by analysing the requested updates against the validity rules presented by the authors. The work is inspired by belief revision principles such as validity, success and minimal change. The authors challenge existing approaches for their lack of completeness and their attempt to address the ontology evolution problem focusing on change operations case by case. They further criticize existing work as error prone, hard coded and giving no guarantee whether the cases are exhaustive. The paper propose a new and different approach to handle ontology evolution by identifying invalidities a given change could cause on the updated ontology using a formal validity model. They further propose an approach to deal with various effects and side effects. The interpretation of effects and side effects is restricted to the validity model which does not differentiate and include semantic effects of change operations. The change implementation process they proposed works under constraints and the constraints come from the validity model.

This work focuses on structural changes and excludes semantic changes which are crucial in ontology evolution. Furthermore, the authors give emphasis to the validity model and exclude other evolution factors such as the user preferences, severity of the change operations and sensitivity of the ontology to $\mathcal{AB}$ox or $\mathcal{TB}$ox statements. An interesting evolution criterion they identify is the minimal change criterion which ensures a minimum number of changes to evolve ontologies.

The major difference between this work and our proposed solution is that, we distinguish structural and semantic impacts, to which we give detailed coverage. Furthermore, our approach focuses on analysing impacts of change operations by focusing on the impacts and the change operations that cause the impacts. Our approach deals with empirically identified impacts and empirically identified atomic change operations that cause the impacts. For determining impacts we focus on a finite set of atomic change operations. This allows us to identify impacts of atomic, composite and domain-specific change operations by combining atomic change impacts and further implementing fine grained change impact analysis. Our approach is capable of identifying impacts of composite and domain-specific change operations.

3.4.2 Consistency Management

Changes in an ontology may introduce inconsistencies in the ontology, in the dependent systems and in the artefacts. Inconsistency management is one of the major focus areas in ontology evolution [Haase et al., 2005] [Haase & Stojanovic, 2005] [Plessers & De Troyer, 2006] [Bell et al., 2007] [Qin & Atluri, 2009]. The two approaches in inconsistency management are the procedural and declarative approaches. The procedural approach maintains consistency by considering the constraints of the consistency model and the definite rules that have to be followed to satisfy them. In this approach, each requested change is checked against a precondition and inconsistency resolution is generated based on a selected evolution strategy. Once the resolutions are generated as additional change operations, the requested change is implemented together with the generated change. The declarative approach maintains consistency by considering a comprehensive set of inferred axioms. This approach treats change requests as changes that must be performed and changes that must not be performed. The consistency of the ontology is checked against the first set of changes and inconsistency resolution is applied by considering both sets of changes to exclude the changes that must not be performed. Finally, the ontology engineer selects one from all the possible consistent states of the ontology [Stojanovic et al., 2002a] [Leenheer & Mens, 2008] [Djedidi & Aufaure, 2010b].

3.4.3 Ontology Change Logging and Mining

Change logging refers to the activity of tracking and recording all changes in a change log during evolution. The change log facilitates recovery of the ontology to its previous state by undoing changes [Leenheer & Mens, 2008]. It is also used to detect changes and discover useful information that can be used later for tasks such as discovering change patterns, co-occurrences of changes and analysis of frequently occurring changes [Javed et al., 2011c].

3.4.4 Ontology Diffs and Content Versioning Systems

Authors in [Noy & Musen, 2002] developed Promptdiff to compare different versions of an ontology. It detects changes in two versions of an ontology and presents the differences. At the end of the evolution process, ontology editors use promptdiff to review changes and approve or reject those changes. Currently, promptdiff does not support OWL2 ontologies. However, there are different successors of promptdiff [Tudorache et al., 2008] [Redmond et al., 2008] that use the heuristics used in promptdiff. [Redmond et al., 2008] suggest a system that manages changes using version control systems. The authors propose a system which addresses the existing problems of ontology version control systems. This includes addressing problems in concurrent editing, complete change tracking, scalability, and performance. They focus on add, delete and rename operations and perform analysis using diffs between two ontology versions.

The authors [Redmond & Noy, 2011] present a pluggable difference engine which aligns ontology entities before conducting comparison. The difference engine uses an alignment phase and explanation phase. The explanation phase organizes the output of the alignment phase and presents the difference in a human understandable and organized way. The difference engine highlights additions, removals and renaming of entities. This approach requires two versions to compare changes. It does not consider the change operations that are the sources of the change.

The authors [Ruiz et al., 2009] propose content CVS (Concurrent Versioning System) for building and editing ontologies collaboratively. They use a CVS paradigm used in software engineering to build ontologies and manage changes. In content CVS the most recent version of the ontology is kept in a shared repository in a server and each developer keeps a local copy. Whenever the developer makes a change to the local copy, he/she has to submit the latest local version to the server. The system compares the request with the most recent version. The developer can access the repository using export, check-out, update and commit operations. If the local version of the ontology is not changed, it means there is no meaningful change committed. Otherwise, if the local version is up-to-date and not in conflict with the recent version, the local version will replace the recent version.

This approach uses change detection, conflict detection and conflict resolution. It uses structural conflict resolution and a combination of structural and semantic conflict resolution. The authors implement deductive difference which computes the logical consequence of the new version with the previous version to identify semantic differences. It uses reasoners to conduct deductive reasoning and semantic conflict refers to the conflict due to inferred axioms. Once the difference is calculated, if there are conflicting axioms or unintended entailments, the users are presented zero or more options to choose. The content CVS allows the user to choose the most suitable minimal plan to avoid the conflicts. If there is no plan, the conflict resolution process ends and the ontology rolls back to the old version.

A closely related work on concurrent development and editing of ontologies is given in [Ruiz et al., 2011]. This work extends content CVS to incorporate several developers to make changes concurrently. This work focuses on conflict detection among change requests from different developers, and resolving the conflicts by employing structural and semantic differences. Semantic conflicts are addressed using logical reasoners.

The authors [Hartung et al., 2012] propose a tool that allows determining semantic changes between two versions of an ontology. A web-based tool, CODEX (COmplex Ontology Diff EXplorer), is proposed. The tool contains a repository for calculating diffs at the backend. The backend computes diffs and presents the changes using statistical measures. This includes: number of changes, diff sizes and growth rates of the changes. It allows exploration of elements that have been influenced by the changes. It further includes change impact analysis to find out the elements that are affected.

CODEX can provide information similar to our change impact analysis tool. However, it follows a similar approach used in diff and in content CVS. Even if we do not follow the ubiquitous diff approach, our change impact analysis tool provides rich analysis and additional semantic impacts other than the semantic changes presented in CODEX. Our approach not only focuses on terminologies, but also analyses impacts on instances and annotation triples. Change impact analysis deals with impacts of changes on annotated documents. It presents impacts of changes on ontology entities and information sources that consume the ontologies.

The authors in [Konev et al., 2012] propose a new version of CEX versioning tool which extends the original CEX [Konev et al., 2008] to incorporate three distinct logical differences. These are: concept inclusion, answers to instance query and answers to conjunctive query. CEX is applicable for acyclic $\mathcal{EL}$ terminologies and the proposed version extends it to ELH+ which admits role inclusion, range and domain restrictions. This enables users to perform concept diff, instance diff and query dif. This work is close to our work by considering semantic changes on instances ($\mathcal{AB}$ox Statements).

In the above approaches, changes are made concurrently and two or more versions of ontologies are compared structurally and/or semantically. Semantic difference focuses on the logical difference of axioms based on inputs from a reasoner. Our approach is different in the following ways. We view impacts from change operations perspectives. First, we focus on the impacts of the change operations that are requested by the user and generated by the system. Second, our notion of structural and semantic impacts is broader than the structural and semantic changes discussed in the above papers. We further incorporate the implication and interpretation of the changes. Third, a minimal plan, in content CVS, refers to a plan that avoids inconsistencies and errors caused by arbitrary entailments with minimum removal of additions or deletions.

Our approach provides detailed information about the requested and the derived change operations, the impacts of the change operations, the affected entities due to a given change operation and the severity of the impact on the entities in the system. Our concern is not only finding the additions and deletions, but also how the entities are impacted, which change operations impact them, how two or more changes impact an entity and the severity of the impacts.

In comparison to the CVS approach, a CVS presents "what" has changed, but the details about how the changes affect the dependent entities, why a given entity is affected and the severity of the effect is missing. Evolution of ontologies using such analysis as an input for selecting an optimal strategy and evolving ontologies is the major concern of this research. At this stage, this research does not analyse impacts of changes on inferred axioms.

3.4.5 Ontology Change Impact Analysis

Change impact analysis is a crucial activity in ontology evolution. Change impact analysis is defined as "the process of identifying potential consequences (side effects) of a change, estimating the cost of implementing the change and analysing alternatives to realize the change" [Bohner, 2002]. The change impact analysis process provides information related to the effects of the required changes in the ontology, other related ontologies and dependent systems that use the ontologies. It is also used to estimate the cost and effort required to implement the requested change [Leenheer & Mens, 2008] and serves as an input for deciding whether to proceed to implement the requested change.

Change impact analysis uses requested changes and dependencies in the evolving ontology, dependent ontologies and artefacts to generate cascaded effects. The impact analysis process includes structural and semantic impacts on all dependent artefacts and should present them for the ontology engineer. The change impact analysis tool can be combined with different evolution strategies to present different options and allows the ontology engineer to choose the strategy which generates less impact and less cost of evolution.

Change impact can be viewed as a normal evolution that preserves existing knowledge according to ontological continuity principle or a revolution that changes existing true axioms. According to [Klein, 2004] [Xuan et al., 2006], change impacts depend on the user's requirement about what to preserve in the ontology. The user may require preserving data instances ($\mathcal{AB}$ox statements) ontology concepts ($\mathcal{TB}$ox statements), inferred facts or the consistency of the overall system. This shows that the analysis and resolution of change impacts is subject to the views and requirements of the user.

Analysis of effects of changes involves checking one or more of the above requirements. It further involves maintenance of inconsistencies by proposing additional changes that address the inconsistencies. This involves a manual procedure where ontology engineers revise the ontology using ontology editors and reasoners to pinpoint the sources of inconsistencies [Stojanovic et al., 2002b].

Optimization of ontology evolution and optimal selection of evolution strategies is given a little attention in the state-of-the-art literature. [Zhang et al., 2008] conduct a study on user defined ontology change and propose an optimization strategy to reduce the time of execution of changes. Their methodology focuses on eliminating redundant atomic change operations. Using redundancy elimination, their methodology optimizes the change implementation in terms of time by reducing the number of change operations. This research explores a new area in ontology evolution covering optimal strategy selection using quantitative analysis of parameters.

To summarize the literature review and to position our proposed research in the context of existing literature, we organized existing research in the following diagram (Figure 3.1). The diagram presents the existing work in two dimensions. The horizontal dimension begins with research approach on pure ontology evolution. Then, as we move to the right, it shifts to research that includes annotations and OCMS evolution. In this dimension the approaches are unified to address the problem of evolution of an OCMS. Relative to existing work, our research approach is built up on previous research and further pushes the boundary to address the evolution problems in an OCMS.

The vertical dimension begins from basic ontology evolution problem. As we go up, this dimension focuses on concurrent evolution, consistency and validity of evolution, and change impact analysis. There are existing researches focusing on addressing these problems in pure ontology evolution context. However, different researchers suggest the change impact analysis approach for ontology evolution. This research explores change impact analysis as an extension of the existing research. It further focuses on proposing methods

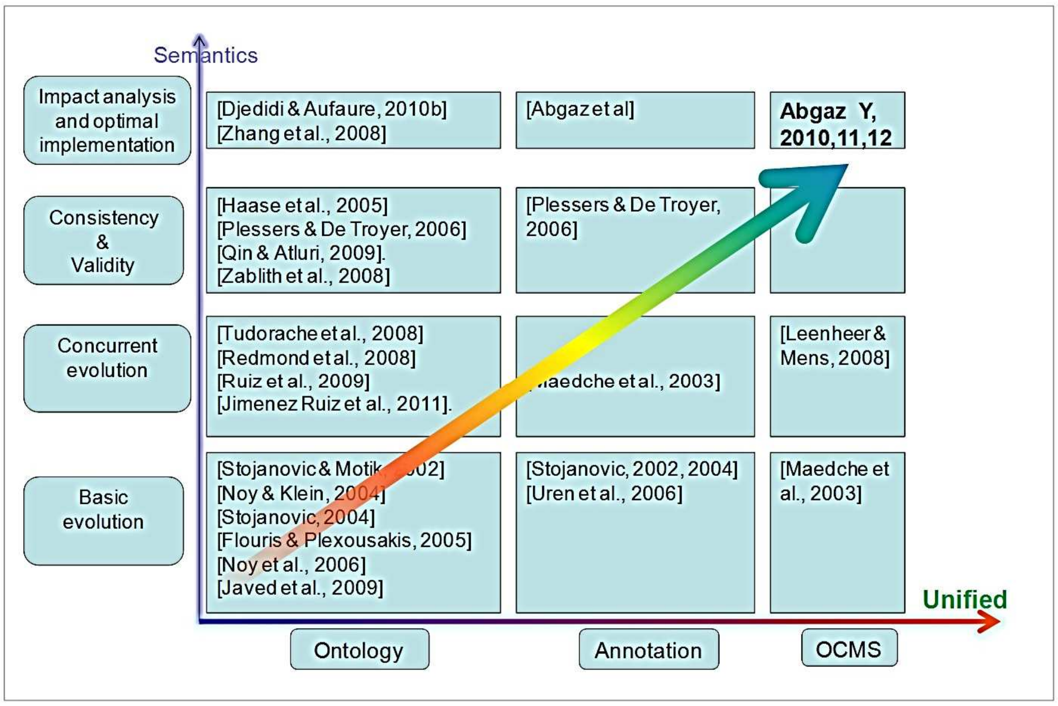

Semantics
Impact analysis and optimal implementation
Consistency & Validity
Concurrent evolution
Basic evolution
[Djedidi & Aufaure, 2010b] [Zhang et al., 2008]
[Abgaz et al]
Abgaz Y, 2010,11,12
[Haase et al., 2005] [Plessers & De Troyer, 2006] [Qin & Atluri, 2009]. [Zablith et al., 2008]
[Plessers & De Troyer, 2006]
[Tudorache et al., 2008] [Redmond et al., 2008] [Ruiz et al., 2009] [Jimenez Ruiz et al., 2011].
[Maedche et al., 2003]
[Leenheer & Mens, 2008]
[Stojanovic & Motik, 2002] [Noy & Klein, 2004] [Stojanovic, 2004] [Flouris & Plexousakis, 2005] [Noy et al., 2006] [Javed et al., 2009]
[Stojanovic, 2002, 2004] [Uren et al., 2006]
[Maedche et al., 2003]
Unified
Ontology
Annotation
OCMS

Figure 3.1: A diagram summarizing existing research

for change impact analysis in a unified but centralized context.

3.5 Tools for Ontology Evolution

There are different tools (Section 2.5) available for supporting ontology evolution. These tools are designed to implement different ontology evolution approaches proposed by researchers. The KAON ontology editor provides support for ontology evolution based on the proposed evolution strategy [Stojanovic, 2004] [Maedche et al., 2003]. The KAON ontology editor allows the user to select different strategies to follow to resolve inconsistencies. The user can configure those strategies before the changes are implemented. The KAON editor shows the intermediate changes but does not cover explicit change impact analysis.

Another tool that supports ontology evolution is the protege editor. The protege editor allows the user to specify changes using the protege user interface. Whenever the user requests a change, the editor asks if the user wants to implement only the requested change or cascade the change to other dependent entities in the ontology. The protege editor has different plug-ins that provide functionalities like change logging, undo and redo services. The main protege editor does not show the affected entities before a change is implemented. However, it provides reasoners that allow the user to check the consistency of the ontology after the changes are applied. It allows the user to undo the changes if the ontology becomes inconsistent.

The NeON editor provides a graphical user interface for evolving ontologies. The NeON toolkit allows the user to choose among three change implementation strategies and allows the user to approve or ignore the changes. It furthermore shows the entities that should be removed or added but does not give information about how the change operation affects the entities and the overall ontology. PromptDiff [Noy & Musen, 2002] [Noy & Klein, 2004] allows a comparison of two versions on ontology and lists all the differences between the versions. PromptDiff requires two versions of an ontology and does not consider the change operations that evolve the ontology from one version to another version. PromptDiff is also available as a plug-in in protege.

3.6 Summary

Existing research covers different problems related to schema evolution, software evolution, ontology evolution and content evolution. Research in the area of ontology evolution covers change specification, change representation and consistency management. It further focuses on change logging, change discovery, change detection, etc.

However, change representation amenable for impact analysis, impact analysis of change operations in evolving OCMS and optimal strategy selection are not yet addressed by existing research. Change implementation in terms of impacts of change operations, severity of impacts, type of statements affected and number of change operations are not sufficiently covered in existing research. However, an OCMS requires efficient management of the evolution of the overall system. This includes analysis of structural and semantic impacts of changes, analysis of impacts of changes using different scenarios (what-if analysis) and selection of optimal implementation strategies using different criteria.

To fulfil these requirements, we propose a layered operator framework that represents changes using different levels of composition; a change impact analysis approach that analyses semantic and structural impacts and an optimal strategy selection method that performs optimal strategy selection by reducing side effects of the change on the OCMS.

Chapter 4

Ontology-based Content Management Framework

4.1 Introduction

Chapter 3 focused on a review of related literature in ontology-based content management systems (OCMS). In this chapter, we present a layered OCMS framework. The conceptual framework is organized into three layers: the ontology layer, the content layer and the annotation layer. We discuss the constructs, the changes and the evolving entities of the layers. A formal representation of the OCMS and its layers using a graph-based formalism is presented.

We further introduce a layered operator framework. The framework contains four layers organized as atomic change, composite change, domain-specific change and abstract changes. The change operations at each layer are represented based upon the graph-based formalism used to represent the OCMS. The two frameworks are used to represent the context in which the change impact analysis is defined and used. They further specify the interactions between changing entities and the interdependences within and across the layers.

This chapter is organized as follows. Section 4.2 provides a general introduction of an

OCMS. Section 4.3 presents the layered OCMS framework and discusses the individual constructs of the framework. The representation of the layered framework using graphs is presented in Section 4.4. Section 4.5 presents the change operator Framework and its formalization. We give an evaluation of the layered operator framework in Section 4.6. Finally we present the summary of the chapter in Section 4.7.

4.2 Ontology-based Content Management System

Content management systems shift toward the use of ontologies to enrich their content and provide a better support for developers, designers and end users [Chu et al., 2009]. In such systems, ontologies facilitate a common understanding and interpretation of a shared knowledge between humans [Gruber, 1993]. However, the use of ontologies is not restricted to the exchange of semantic information between humans and computers. It further transcends that and incorporates the exchange of semantic information among autonomous digital devices. This is achieved by annotating the target content using ontologies in such a way that both human and computer systems gain the same understanding of the semantic meaning conveyed by the content.

The semantic information, which is available in OCMS, is used for different purposes. It is used for creating taxonomically guided information organization [Jones et al., 2011], discovering previously unknown information, conducting semantically assisted information retrieval and so on [Vallet et al., 2005] [Jun-feng et al., 2005]. It is used to identify more relevant documents to the user's query. It is used to improve the precision of the results by filtering content which is not relevant to the information need [Navigli & Velardi, 2003] [Paralic & Kostial, 2003].

The semantic information is used to classify content using taxonomies and hierarchies [Fernández et al., 2011]. Semantically rich content is used to identify hidden relationships between the content, authors, publications, etc. [Pahl et al., 2010]. In other application areas such as social networks, this semantic information is used to analyse activities of users, product preferences, patterns of usages, etc. [Mika, 2007].

To achieve this, OCMSs make use of both generic and domain-specific ontologies. The generic ontologies provide semantics for concepts whose interpretation is not restricted to a specific domain. Examples of such ontologies are SUMO and MILO[1] which give semantic information about countries, measurement units, currencies and so on. Despite their applicability to a wide range of disciplines, generic ontologies do not provide the detailed semantics required to describe content in specific domains. Content-based systems, that are built to support specific domains, make use of ontologies that are specifically built for those domains. This enables the content-based system to supplement the content with rich semantics using domain-specific ontologies.

Thus, the primary objective of OCMSs is to make the content understandable, interpretable and interoperable by both humans and machines based on a common specification and representation of concepts in a given domain.

An OCMS resides either on the web or on a private network. It may use a standalone system where every required component is stored in one place or may utilize a distributed environment where all the components or parts of components come from different locations. Furthermore, the authors of the content may use one or more ontology to describe the content. Semantic annotation of content is also done in different ways. Authors of the content may use in-line annotation which embeds the annotations in the original content or a stand-off annotation which stores the annotation separately from the original content.

We built the OCMS framework to have a common understanding of what we mean by an OCMS. The OCMS framework defines the different layers of a typical OCMS, their interaction and the conceptual location of each layer. Our framework supports the following OCMS requirements. The framework needs to:

- allow extensibility of any of the layers namely the ontology, the annotation and the content layers.
- ensure a transparent interaction of the layers and clear dependencies between them.
- provide a method for simple evolution of the individual components.

[1] http://www.ontologyportal.org/

- facilitate change implementation with minimum or predictable impact on the dependent entities.

- maintain the consistency within and across the layers.

To realize these requirements, we propose a layered OCMS framework with three distinct layers. The interaction between the layers is defined by the dependencies that exist among the entities in the layers. The framework facilitates the evolution of the components with a visible and transparent effect on the dependent layers. The framework further maintains the consistency of the system whenever there is a change in any of the components. The following subsection introduces the different components of the OCMS framework.

4.3 Layered OCMS Framework

The first layer of the OCMS framework is the ontology layer. The annotation layer is the second layer which contains annotation triples. The third layer is the content layer which contains a set of documents. The framework is depicted in Figure 4.1.

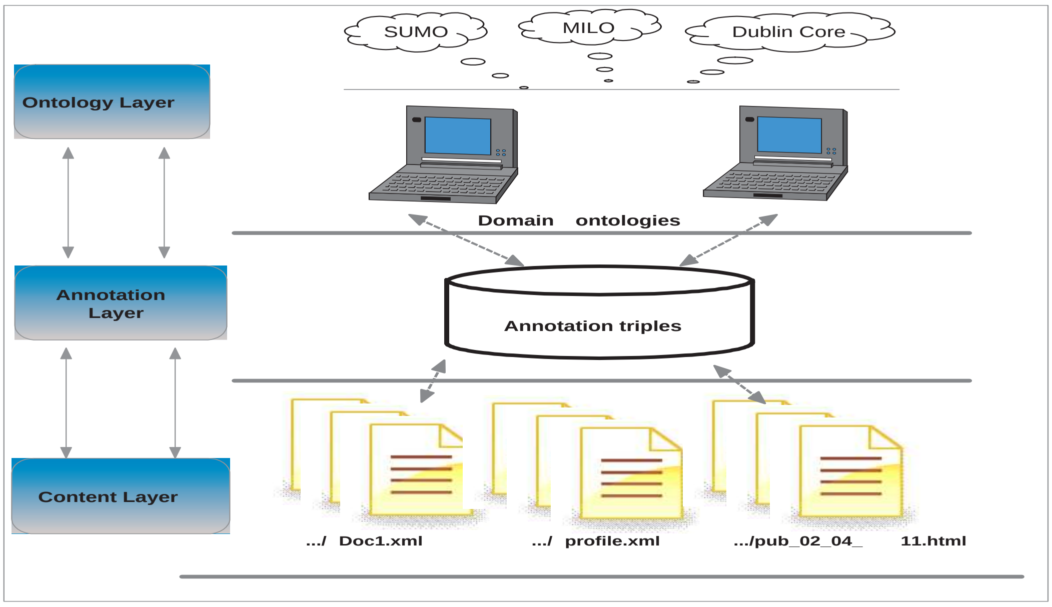


Figure 4.1: Layered framework of OCMS

4.3.1 Ontology Layer

The ontology layer is a layer which contains one or more ontologies that provide semantics to a given content in the OCMS. Ontologies provide a common ground for understanding, conceptualization, representation and interpretation of domain concepts uniformly across different systems, languages and formats. Researchers [Guarino, 1998, p.1] further explained an ontology as:

"An engineering artefact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words"

Our OCMS framework allows the use of one or more ontologies to describe content. This is achieved by exploiting URIs to uniquely identify the ontologies. Ontologies are widely represented using web ontology language (OWL)[2]. We select OWL due to its expressiveness to represent domain-specific knowledge. Furthermore, it is the recommendation of the W3C, widely used in many ontology applications and is expressive along with a formal semantics [Grau et al., 2008] [Ardil, 2005]. We can achieve maximum interoperability of OCMS with other ontology-based applications by using OWL.

Single ontology versus multiple ontologies. There is no single ontology that covers every concept defined in the world. However, users generate content that provides deep coverage of a single discipline or shallow coverage of multiple disciplines. Others may cover anything in between the two extremes. The ontology layer should support users to benefit from both domain-specific and generic ontologies.

Thus, in a single ontology environment, the OCMS uses one ontology to describe everything in the content. A good example is the gene ontology[3] used in medical sciences which represents a single domain. However, there are other applications that use multiple ontologies. For example, the Semantic Web for Earth and Environmental Terminology (SWEET)[4] ontology merges several modular ontologies together to describe different con-

[2]http://www.w3.org/TR/owl2-primer/
[3]http://www.geneontology.org/
[4]http://sweet.jpl.nasa.gov/ontology/

cepts from different disciplines.

Generic versus domain-specific ontologies. The ontology layer needs to support both domain-specific and generic ontologies. The support of domain-specific ontologies can be used for detailed and precise annotation of domain-specific content. Generic ontologies that cover concepts which are generic such as space, time and measurement, etc., and can be used to provide shallow annotation of a generic content.

Standalone versus distributed ontologies. Ontologies in the ontology layer can be stored together on a single machine or distributed on different machines on the web. When publicly available ontologies are used in standalone environments, a local copy of the ontologies is maintained, however, it is also common to use ontologies distributed over different sites.

A choice has to be made before the implementation of an OCMS, because these factors determine the behaviour of the overall OCMS.

4.3.1.1 Changes in Ontologies

Ontologies change frequently and continuously throughout a life cycle of an OCMS. Whenever there is a change in the domain, its conceptualization or specification, related ontologies need to evolve [Noy & Klein, 2004] [Benjamins et al., 2002]. When new concepts are added, existing ones are deleted or modified in the content, the respective ontology needs to be updated.

According to research conducted by [Goncalves et al., 2011] on a biomedical ontology of the National Cancer Institute (NCI), the most frequent changes in an ontology focus on addition of new classes, renaming of existing classes and addition and deletion of subclass axioms. Whenever there is a substantial change in ontologies, the deletion of classes, data properties and object properties are frequently observed.

Other research [Curino et al., 2008] identifies the intensity of the change in Wikipedia's database schema, which is the best-known example of a large family of web information systems (WIS). They identified 170+ documented schema versions over 4.5 years and over 700GB of data and more than 88,397 revisions in Mediawiki in 2007. They reported the growing frequency of change as “There is strong pressure toward change (from 39% to 500% more intense than the traditional setting)”.

Implementing the changes requires understanding them correctly and representing them accurately using change operations. However, this only solves few of the associated problems. These changes can trigger further cascaded changes and affect one or more interrelated ontologies. The impacts of the change may propagate to instances leaving the process in a vicious circle. An ontology engineer who detects a change of an instance in a content document and tries to maintain the ontology accordingly may end up with many unseen and unexpected changes. Furthermore, when the size of the ontology is large, it becomes difficult to understand and trace the propagation of the change and its impact on other entities in the ontology and on dependent systems. The manual management of changes, in this case, does not ensure the complete identification of the impacts and impacted entities. It does not guarantee the implementation of the requested change without introducing additional changes which are not initially identified by the user.

4.3.2 Content Layer

The content layer contains content documents which are the subjects of semantic annotation [Kiryakov et al., 2004]. We define content as any digital information which is in a textual format that contains structured or semi-structured documents, web pages, executable files, software help files, etc. An OCMS essentially deals with content in the form of books, web pages, blogs, news papers, software products, documentations, help file reports, publications, etc. [Gruhn et al., 1995] [Abgaz et al., 2010]. The content layer provides the following services.

Storage of documents. The content layer facilitates the storage of content. The storage can be file-base or database storage. The content layer stores content as files in folders or tables in databases and are accessible from the web. In any of the two cases the content layer provides a permanent storage of the content.

Retrieval of the documents. Another service provided by the content layer is the retrieval of the documents whenever users require them [Kiryakov et al., 2004]. The retrieval service provided by the content layer is crufcial in accessing a specific document.

Unique identifiers for content. All documents and parts of documents should be identified uniquely. The unique identifiers serve as content identifiers to link the content with the ontology. Documents in the file-based storage are identified using the path and the file names. However, in databases they are identified using the database name, the table name and the primary key [Elmasri & Navathe, 2010]. Documents that are stored on the web can be accessed using the URI of the web combined with the file names. However, the detailed implementation is the decision of the content manager.

Content in OCMS can be categorized as structured content, semi-structured content or unstructured content.

Structured content. Structured content is content which is well defined with respect to some data-centric structure. Data-centric structure defines the content or fragments of the content as data elements with a schema describing the elements. The content gets the structure by explicitly tagging parts of the content with the schema. A widely used format is XML. XML is supplemented by DTDs and XML schema (Section 2.2) to provide further semantics about the data. In a structured document, it is possible to locate and retrieve a specific part of a document using the data elements. The other widely used format for structured content is databases. Relational databases store the content in the form of tables which are organized into columns and rows. The rows represent individual instances and the columns represent the attributes of the instances. The content or part of the content in databases is accessible using queries which extract specific rows and columns [Elmasri & Navathe, 2010].

In structured documents, there is interdependence between parts of the content documents using tags/attributes that allow composition of new content from the available snippets. Since the fragments of content are highly structured and identifiable using content identifiers, it is possible to combine different content fragments into one and present that as a new content fragment. Such relation between different content can be identified using structures such as DocBook. DocBook[5] defines the logical structure of a document in the

[5]http://www.docbook.org/

form of XML, HTML, XHTML, etc., (Section 2.2).

Semi-structured content. Semi-structured content is content which is organized using a document-centric structure. A document-centric structure gives structure to the whole document or part of the document focusing on its presentation. In such documents, it is possible to access information based on the available structure, but it needs additional effort to locate and retrieve specific data elements. Content in an HTML file can be considered as semi-structured content which incorporates tags that give some structure to the presentation of the content.

Unstructured content. Unstructured content refers to content which does not have any structure defined for identifying components of the content. Unstructured content holds a series of texts where there is no associated structural information that gives the content a structure.

Our OCMS layer allows any kind of content to be annotated including unstructured content. However, for the purpose of this research, we focus only on content which is either structured or semi-structured.

4.3.2.1 Changes in the Content

Content in OCMS evolves continuously and frequently [Uren et al., 2006] [Adler et al., 2008] [Krotzsch et al., 2011]. The evolution may cause a change in the semantics or in the structure of the content. Changes that affect the structure also cause a change in the semantics. In a dynamic content management system new documents are produced, existing ones are modified, edited or deleted frequently to provide up-to-date information. The content layer allows changes ranging from removal of the whole document to modification of a single element in the document. The changes of the content in the content layer need to be available to the other layers to ensure the consistency of the OCMS system [Javed et al., 2010].

We focus on structured and semi-structured content in the content layer. This is to avoid complications related to accessing and processing changes in unstructured documents. Structured and semi-structured content further allow us to easily identify evolving elements of the content and create a unique reference which can be used for later processing. Thus, in this research, we primarily focus on XML and HTML content documents. The content documents have associated URIs. In case of XML documents, the different sections are identified by combining the URI with the element ID. Whereas, in HTML and XHTML files, we identify specific parts of the content with an offset showing the beginning and the end of the section relative to the document [Maynard, 2008].

4.3.3 Annotation Layer

The annotation layer provides a means of handling semantic annotation [Chu et al., 2009]. Semantic annotation is a process of linking content with ontology entities to enrich the content with the semantics [Oren et al., 2006]. Semantic annotation is used to explicitly identify concepts and relationships between concepts discussed in the content [Uren et al., 2006] [Krötzsch et al., 2007].

The annotation process semantically enriches content by defining its attributes using concepts and properties from the ontology. It further creates a link between two content documents to indicate their semantic relationships. Annotation also refers to the output of the annotation process.

In any application that makes use of ontologies, the target content, which needs to be semantically enriched, is required to have an explicit link, at least to one or more elements in the ontology. The annotation becomes the major element of the OCMS for the following reasons.

- In applications that make use of ontologies, the target content which needs to be semantically enriched is required to have an explicit link at least to one element in the ontology. This is achieved by annotations.
- Annotation provides semantics which can be used by humans and machines.
- Annotation provides traceability of the content fragment using the semantics associated with it.

4.3.3.1 Annotation Storage

In semantic annotation, there are two approaches commonly used to store annotation data: in-line annotation and stand-off annotation [Wilcock, 2009]. The in-line approach embeds the annotation information in the content. Such annotations either modify the content of the original document to embed the annotation or maintain a copy of the original document together with the annotation data. Whenever the semantic data is required, the system needs to access the annotated document and extract the annotations[6]. The disadvantage of the in-line annotation is that the annotation must be aligned with the $\mathcal{TB}$ox statements which requires additional effort to align the content with the ontology.

The stand-off annotation stores the annotations of the document in a separate storage space. This approach uses the document URI as a unique identifier of the documents and every annotation of that document is associated with a URI. This approach has advantages and disadvantages. The first advantage is the separation of the semantics from the content, which allows independent evolution of either the content or the annotation. The second is, it enables the annotation data to be accessed separately without reading the whole document. Exhaustive annotation increases the size of the original document and becomes a problem for accessibility of individual annotations [Maynard, 2008]. Third, it is suitable to annotate content when the annotator does not have the permission to modify the content. The separate annotation layer further provides facilities such as querying the annotation triples. However, there are disadvantages associated with it [Wilcock, 2009].

The main disadvantage is, it requires a systematic synchronization of the annotation with the content. When the document is modified or deleted, the annotation layer should be updated accordingly. In a distributed environment, this task may introduce additional overhead. The other disadvantage is the separate storage of the content and the annotation. The separation causes the content to get delivered separate from the annotation. In fact, this problem can be addressed by merging the content and the annotation data during content delivery.

[6]http://www.w3.org/TR/sawsdl/#Using

Table 4.1: Annotation triple representation

Subject	Predicate	Object	context
CNGL:id-2.xml	rdf:type	rdfs:Resource	cngl:triple
CNGL:id-2.xml	rdf:type	CNGL:Document	cngl:triple
CNGL:id-2.xml	CNGL:isAbout	CNGL:DeletingEmail	cngl:triple
CNGL:id-2.xml	CNGL:hasTitle	“Deleting email account”	cngl:triple
CNGL:id-2.xml	CNGL:Contains	CNGL:id-6.xml	cngl:triple
CNGL:id-2.xml	CNGL:mediaType	CNGL:Text	cngl:triple

4.3.3.2 Annotation Triples

The annotation process uses RDF triples (subject, predicate and object) to annotate any content document. It further stores the context of the annotation to distinguish between different contexts. The subject of the annotation comes from the content layer and is usually a URI. The predicate comes from the ontology or the schema defined for the ontologies. The objects can be resources from the ontology or other content artefacts. A single content document can have multiple annotation triples. A single resource defined in the ontology can be used many times in the annotation layer.

Table 4.1 shows the structure of the annotation triple. The subjects of the annotation are content documents (xml files in this example) or parts of xml files. The predicates originate from OWL or RDF properties (rdf:Type) or properties from domain-specific ontologies (CNGL:isAbout). The objects come from either the ontology (CNGL:Document) or from the content layer (CNGL:id-6.xml). The user can provide different contexts to categories of annotation triples. For example, the context of the triples is CNGL:triple, to indicate that they are triples for annotating resources in CNGL.

In the OCMS, we store the annotation triples in triple stores. Triple stores improve the speed of the retrieval of the required information. They store large numbers of triples and are suitable for further expansion [Bizer & Schultz, 2008]. Furthermore, the annotation triples are compliant to RDF and RDF/XML serializations.

4.3.3.3 Change in the Annotation

The annotation layer is the dynamic layer of an OCMS [Goncalves et al., 2011]. The changes in the annotation layer are frequent and include addition and deletion of individual annotations. There are a number of triples added, modified or deleted in this layer. This layer is highly dependent on both the content and the ontology layer. Any change in the other two layers affect the annotation layer which carries all the semantics related to the content. Changes made on the triples of this layer may cause other changes to related annotations within the layer. In such a situation, the changes in the annotation layer require proper analysis and evaluation before they are implemented in the system.

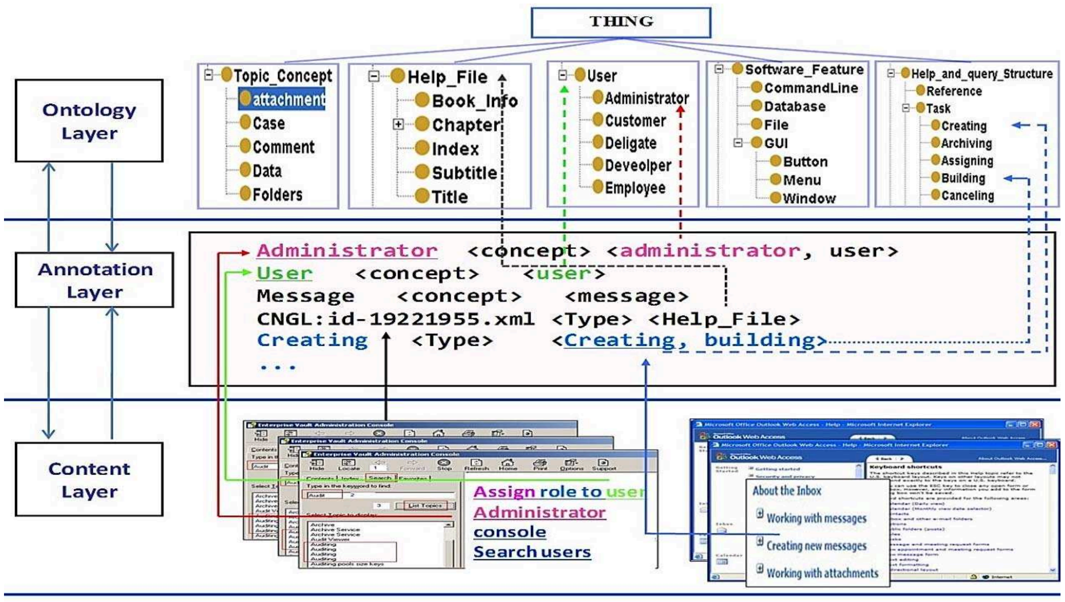


Figure 4.2: An example of a layered framework of OCMS

Now, let us take a concrete OCMS representation and see how the three layers interact with each other. In Figure 4.2, the ontology layer contains an ontology which contains concepts such as *Help_file, User, Software_Feature*, etc. These concepts are used in the annotation layer to describe help documents stored in the content layer. For example, the document that contains information about an administrator is linked to the concept *administrator*. Another document is linked to the task of creating and building.

4.4 Graph-based Representation of an OCMS

The OCMS is represented using a graph-based formalism. We choose a graph-based formalization over set theory or relational algebra for the following reasons. First, graphs provide exhaustive theory support and reduce the problem to a well studied topic in graph theory [Baresi & Heckel, 2002]. This includes mappings between structures and finding a minimal representation of a given graph. In this research, we frequently search entities in the OCMS to delete or add semantics. Graphs have some proven efficiency for searching subgraphs, nodes and edges. There are generic implementations and algorithms available for graphs [Heckel, 2006].

Second, graphs provide appropriate data structure to represent ontologies and annotations. The available ontology editors, such as protege, use graphs to represent ontologies in RDF and OWL [Trinkunas & Vasilecas, 2007] [Bönström et al., 2003]. Finally graphs visualize complex data in a simple and understandable way. In our OCMS, the ontology and the annotation are represented as graphs and the content is represented as a set of documents. The document set serves as a node (of type instances) in the annotation layer.

An OCMS is represented as graph $G = G_o \cup G_a \cup Cont$, where G_o is the ontology graph, G_a is the annotation graph and $Cont$ is the content set. An example of a graph representation of an OCMS is given in Figure 4.3 representing the ontology graph at the top, annotation graph in the middle and the document set at the bottom. Each of the individual graphs and their descriptions are given below.

4.4.1 Ontology Graph

An ***Ontology Graph*** is represented by a directed labelled graph $G_o = (N_o, E_o)$ where N_o is a set of labelled nodes $n_{o1}, n_{o2}, \ldots, n_{ol}$ which represent classes, data properties, object properties and instances [Zhang et al., 2010]. E_o is a set of labelled edges $e_{o1}, e_{o2}, \ldots, e_{om}$. An edge e_o is written as (n_1, α, n_2) where $n_1, n_2 \in N_o$ and the labels of an edge represented by $\alpha \in CA \cup DPA \cup OPA \cup IA \cup RA$.

CA={subClassOf, disjointClasses, equivalentClasses}

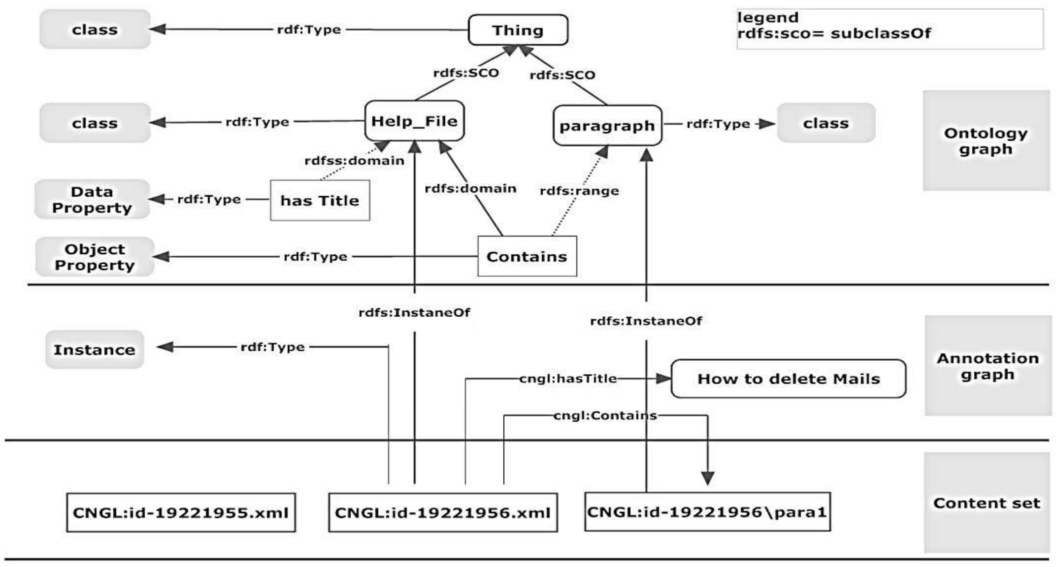


Figure 4.3: Graph-based representation of OCMS

DPA={subDataPropertyOf, dataPropertyRange, dataPropertyDomain, disjointDataProperties, equivalentDataProperties, functionalDataProperty}

OPA={subObjectPropertyOf, objectPropertyRange, objectPropertyDomain, disjointObjectProperties, equivalentObjectProperties, inverseObjectProperties, symmerticObjectProperties, functionalObjectProperty, inverseFunctionalObjectProperties, transitiveObjectProperty, reflexiveObjectProperty, irreflexiveObjectProperty}

IA={sameIndividual, differentIndividuals, classAssertion, dataPropertyAssertion, objectPropertyAssertion }

RA={objectAllValuesFrom, objectSomeValuesFrom, objectHasValue, objectHasSelf, objectExactCardinality, objectMaximumCardinality, objectMinimumCardinality, dataAllValuesFrom, dataSomeValuesFrom, dataHasValue, dataExactCardinality, dataMaximumCardinality, dataMinimumCardinality }

In the ontology graph, properties are often represented as nodes and property instances are represented as edges [Bönström et al., 2003]. User defined property nodes link with other class nodes using schema level property instances such as *rdfs:domain* and *rdfs:range*. For example, in Figure 4.3, the object property *Contains* is a property node linked to

Help_File using a schema level instance property *rdfs:domain* as an edge. This schema level property instance defines the domain of the property node. However, at the annotation level, the property which is treated as a node in the ontology (now serving as schema for the annotation graph) is treated as an edge in the annotation instead of a node. For example, an instance of *Help_File CNGL:id-a9221956.xml* is linked to an instance of a paragraph using the edge *cngl:Contains*.

In general, we treat properties as nodes and property instances as edges (Figure 4.6). When we define properties as part of an ontology, we represent them as nodes and when we use those defined properties in the annotation, we represent them as edges of the annotation graph. We define the properties as a node and link them with other class nodes and property nodes in the ontology graph. We represent property instances as edges that link two instances in the annotation graph.

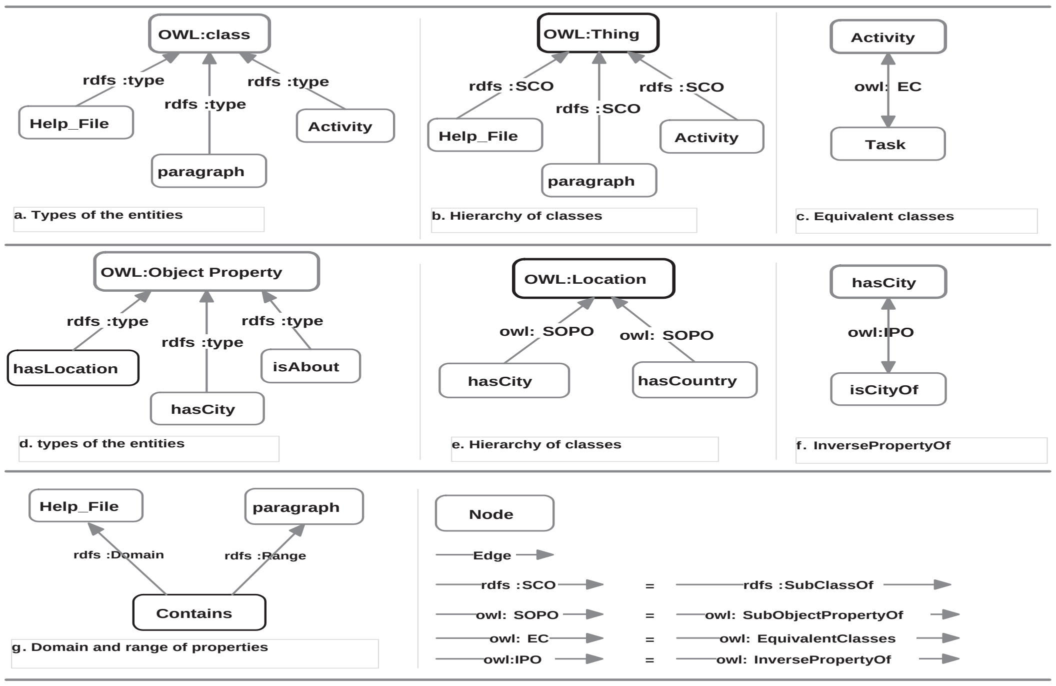
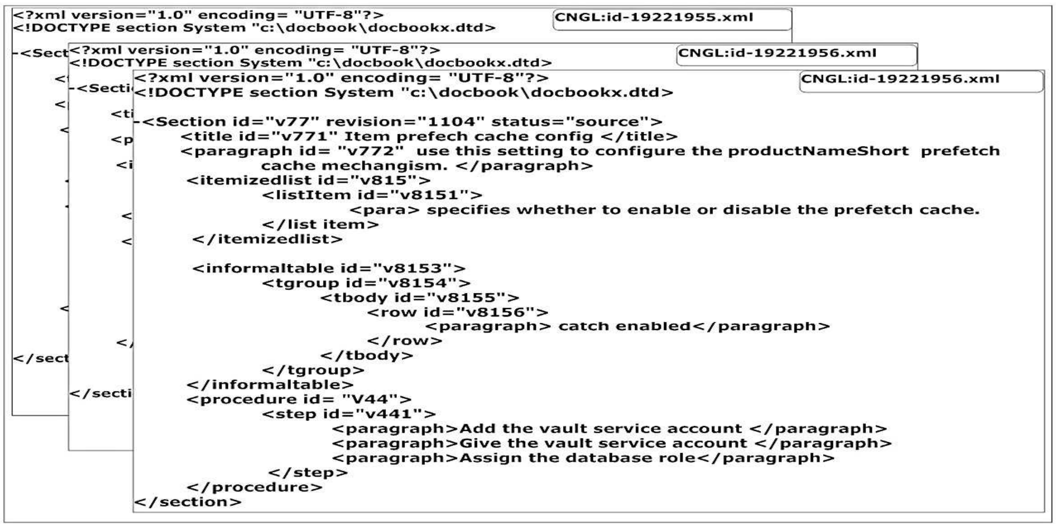


Figure 4.4: Graph-based representation of the ontology layer

In Figure 4.4.a, the graph nodes represent entities that are linked to the owl:class node. Universal classes and domain-specific classes are defined as owl:class. The edge links each of the entities to the owl:class node. Figure 4.4.b shows the relationship among the nodes. The edges represent a subclass axiom. In Figure 4.4.c, the edges represent the equivalence axiom between the two nodes, which are represented as classes in Figure 4.4.a. The representations of the property nodes and edges created between properties are given in Figure 4.4.d, Figure 4.4.e and Figure 4.4.f. Figure 4.4.g represents nodes and edges between classes and properties.

4.4.2 Content Set

A ***Content Set*** can be viewed as a set of content documents. $Cont = \{d_1, d_2, \ldots, d_n\}$ where: d_i represents a structured or semi-structured document or elements of a document. In the content layer, such content is represented as a node.



```
<?xml version="1.0" encoding= "UTF-8"?>                CNGL:id-19221955.xml
<!DOCTYPE section System "c:\docbook\docbookx.dtd>
```

```
<?xml version="1.0" encoding= "UTF-8"?>                CNGL:id-19221956.xml
<!DOCTYPE section System "c:\docbook\docbookx.dtd>
```

```
<?xml version="1.0" encoding= "UTF-8"?>                CNGL:id-19221956.xml
<!DOCTYPE section System "c:\docbook\docbookx.dtd>

-<Section id="v77" revision="1104" status="source">
     <title id="v771" Item prefech cache config </title>
     <paragraph id= "v772"  use this setting to configure the productNameShort  prefetch
               cache mechangism. </paragraph>
      <itemizedlist id="v815">
               <listItem id="v8151">
                         <para> specifies whether to enable or disable the prefetch cache.
               </list item>
       </itemizedlist>

       <informaltable id="v8153">
               <tgroup id="v8154">
                     <tbody id="v8155">
                          <row id="v8156">
                                <paragraph> catch enabled</paragraph>
                          </row>
                     </tbody>
               </tgroup>
      </informaltable>
      <procedure id= "V44">
               <step id="v441">
                      <paragraph>Add the vault service account </paragraph>
                      <paragraph>Give the vault service account </paragraph>
                      <paragraph>Assign the database role</paragraph>
                </step>
      </procedure>
</section>
```

Figure 4.5: Document collection

The content is represented as a set of documents either in a flat file, or in a database. We represent the set of documents using their unique identifiers. The unique identifiers ensure access to the exact location of the documents. However, the selection of storage structure is the decision of the architects at the time of deployment of the OCMS.

4.4.3 Annotation Graph

An ***Annotation Graph*** is represented by a directed labelled graph $G_a = (N_a, E_a)$ where N_a is a set of labelled nodes $n_{a1}, n_{a2}, \ldots, n_{al}$ and E_a is a set of labelled edges $e_{a1}, e_{a2}, \ldots, e_{am}$. An annotation edge e_a is written as $(n_{a1}, \alpha_a, n_{a2})$ where $n_{a1} \in Cont$ is a subject, $n_{a2} \in Cont \cup G_O$ is an object and $\alpha_a \in G_O$ is a predicate. The edges are referred as triples.

The user-defined properties are treated as labels of the edges when they are used in the annotation layer to describe the document nodes. For example, in the triple *(CNGL:id-19221955.xml, cngl:Contains* , CNGL:id-19221955\para) in Figure 4.6, the document is treated as a node and the instance property *contains* is represented as the label of the edge in the annotation layer. Figure 4.6 further depicts the sources of the objects in the triples. The triples in the vertical ovals get their object from the ontology graph, whereas the triple in the horizontal oval gets its object from the content set.

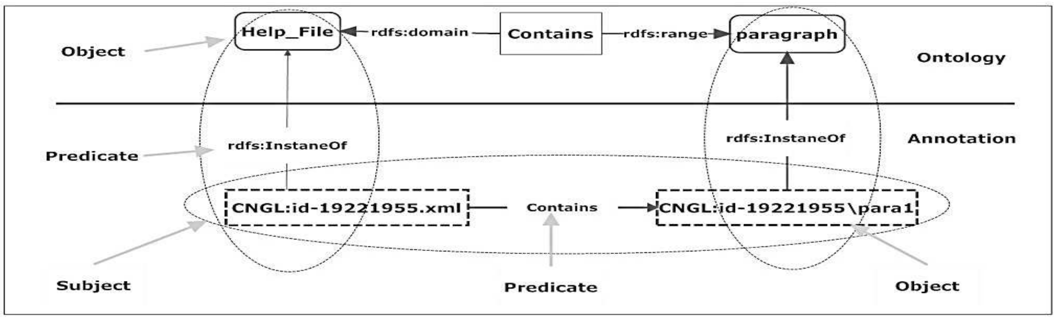


Figure 4.6: Annotation graph

4.4.4 Attributes of the Graph

The type of a node is given by ***type (n)*** that maps the node to its type which is defined in the schema (class, instance, data property, object property). The label of any edge $e = (n_1, \alpha, n_2)$, which is α , is a string given by ***label(e)***. The label of a node n is the URI associated with the node and is given by ***label (n)***. All the edges of a node n are given by a function ***edges (n)***. It returns all the edges as $(n, \alpha, m) \vee (m, \alpha, n)$ where n is the target node and m is any node linked to n via α.

4.5 Change Operator Framework

To process and implement a requested change properly, we need to represent the changes in relation to the OCMS framework (Section 4.3) and its graph-based formalization (Section 4.4). The change representation needs to ensure the correct and complete representation of the requested change. The change request may vary depending on the objective of the user and the size of the desired change. A change request can contain a single task and can be represented by a single change operation, but this is not always true. Change requests may become complex and may not be represented by a single change operation. For such change operations, we need to combine atomic change operations to form a composite change operation. For change requests in domain-specific ontologies, we may be interested to represent changes that have similar patterns. Thus, we represent them using domain-specific change patterns.

Our change representation framework covers three main components. The first component provides a high level conceptual representation of changes using a layered operator framework (Section 4.5.1). The framework specifies the organization of change operations to represent changes in a suitable way. The second component provides a metamodel for representing individual change operations (Section 4.5.2). It provides specification for the attributes of the change such as target entities, parameters, owner, time stamp, order, etc. The third component provides specification of change operations using graph formalizations (Section 4.5.3). This component represents the implementation of the actual change in the OCMS graph and how the individual change operations change the overall system. Each of these components are discussed from Section 4.5.1 to 4.5.3 in detail.

4.5.1 A Framework of Change Operators and Patterns

To represent changes in a suitable format, we propose a layered operator framework which contains four different levels [Javed et al., 2009] [Javed et al., 2011b]. The first level contains elementary changes that represent atomic tasks. The second level contains aggregation of atomic change operations that represent composite and complex tasks. The third level contains a mix of atomic and composite change operations to create domain-specific change patterns. The fourth level focuses on generic categorization of the domain-specific change patterns. The first two levels are considered as generic change operations. The last two levels represent patterns of change operations.

- Level one: elementary changes which are atomic tasks.
- Level two: aggregated changes to represent composite, complex tasks.
- Level three: domain-specific change patterns.
- Level four: abstraction of the domain-specific change patterns.

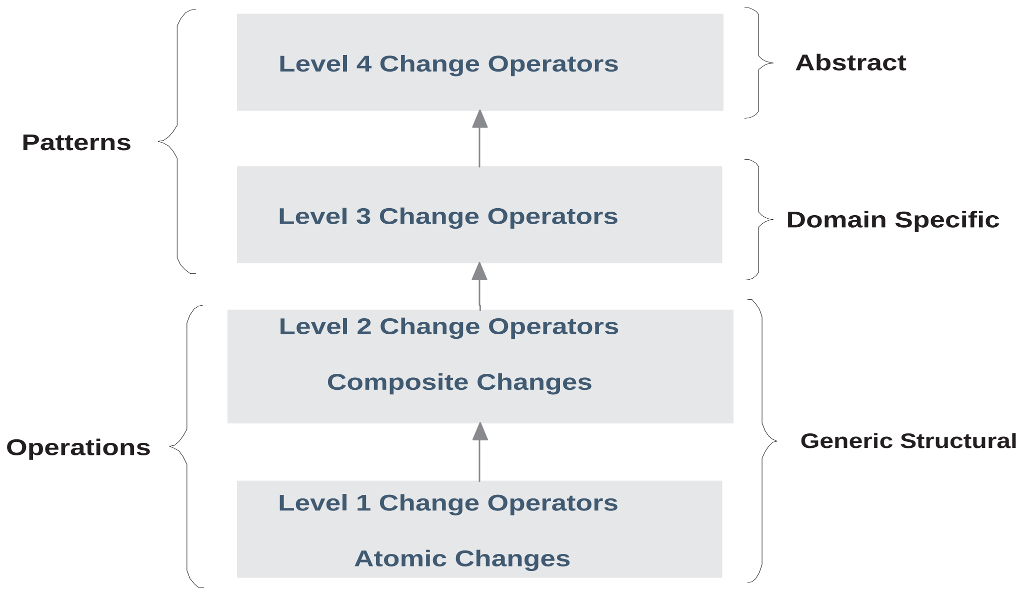


Figure 4.7: Layered operator framework

We observed that [Javed et al., 2009] ontology changes are driven by certain types of common, often frequent changes in the application domain. Therefore, capturing these in the form of common and regularly occurring change patterns creates domain-specific abstractions. A number of basic change patterns may be provided so that users may adapt and generate their own change patterns to meet their own domain-specific demand. This makes the ontology evolution faster and easier.

4.5.1.1 Generic Structural Levels

Level One Change Operators - Element Changes. These change operators are the elementary operations used to perform a single task by an ontology management tool. These operators add or remove a single entity in the ontology. A single operator performs a single task that can add or delete a single concept, a single property, etc. Level three and level four change patterns specified in Figure 4.8 are based on patterns observed in a university administration domain (Appendix C) and can be abstracted to other domains that use similar patterns.

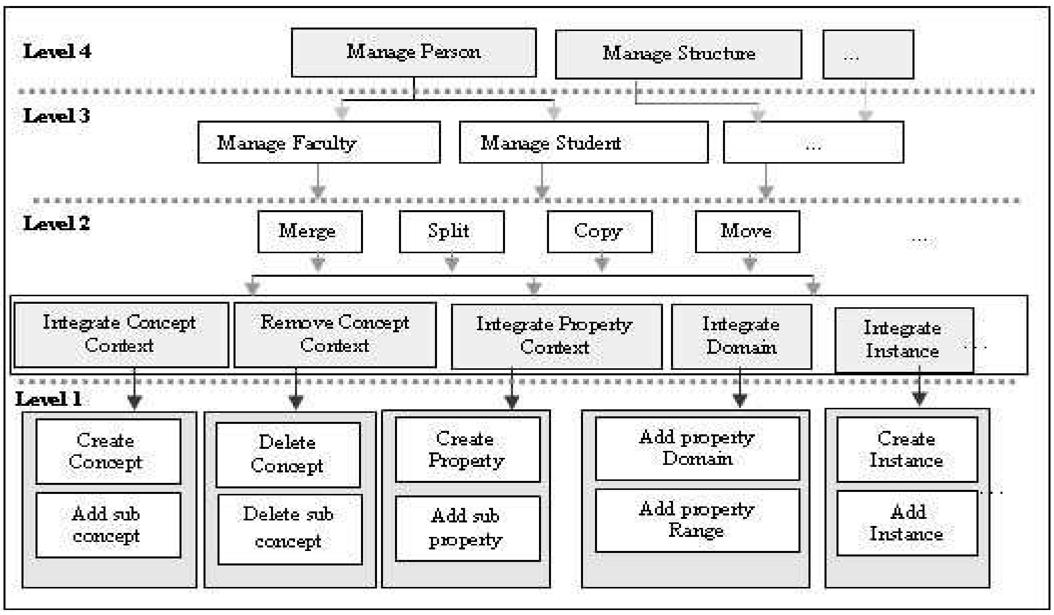


Figure 4.8: A layered operator framework - detailed view

Level Two Change Operators - Element Context Changes. Many evolution tasks cannot be done by a single atomic change operation. A set of related change operations is required. These change operations are identified by grouping atomic operations to perform a composite task. Users request to implement changes that cannot be done by a single atomic change. Such changes are composition of atomic change operations in a specific order. The combination of the atomic change operations are determined by the type of the requested change. It is possible to create an infinite number of composite change operations from the atomic change operations. For example, when a user wants to split an existing concept into two distinct concepts, he/she combines atomic change operations in a specific order to represent the requested changes. A single composite change operation can be represented in many ways using different combinations of atomic change operations. In this research, we focus on supporting composite change operations. We further provide predefined and frequently used composite change operations such as merge, split and copy [Stojanovic, 2004] [Javed et al., 2011a] [Javed et al., 2012].

4.5.1.2 Domain-Specific Level

Level Three Change Operators - Domain-specific. This domain-specific perspective links the structural changes to the aspects represented in domain ontologies. In order to execute a single domain-specific change, operations at level two are used. The change patterns are based on the viewpoints and preferences of the users. Two users may have different perspectives to view the ontology, which results in the use of a combination of different operations from composite changes. As the perspectives are different, the number of operations or the sequence of operations may differ. This difference results in patterns of changes based on the perspectives of the ontology engineers. Domain-specific change patterns are extracted from change logs over a long period of time that represents the patterns of change operations used to implement changes. The extraction of the patterns from the change logs are discussed in detail in [Javed et al., 2011a]

Level three operators enable us to treat domain-specific operations separately and allow us to define our own change patterns once and execute them many times.

4.5.1.3 Abstract Level

Level Four Change Operators - Generic Categorization. Level four change operators are constructed based on the abstraction of the concepts in level three. The main objective of introducing this level is to provide a facility that maps domain-specific ontologies to available upper level ontologies (i.e. categorizing domain concepts in terms of abstract ones) and to generalize and transfer patterns to other domains. Level four is considered as a framework aspect that guides the transfer of patterns to other domains. It is not directly available for operational implementation. It provides abstract mapping of change patterns used in one domain to similar change patterns required in another domain.

4.5.2 Change Metamodel

Following the layered operator framework, we identify and represent changes and their attributes using a metamodel. Whenever a change operation is executed, we store the change operations which are additions and deletions of classes, data properties, object properties, instances, axioms, etc. The model captures information about the change operations. This information is useful to handle composite changes and domain-specific patterns. The metamodel of a change is given in Figure 4.9 and an example of an atomic change operation is depicted in Figure 4.10.

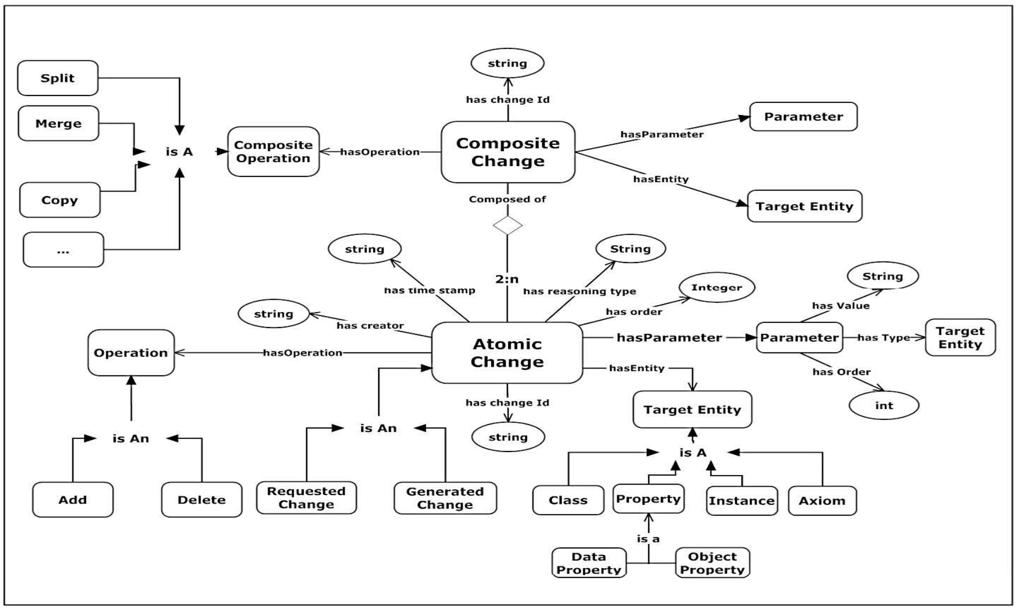


Figure 4.9: A metadata model for change operations

4.5.2.1 Change

Atomic change represents a single change operation which performs a single task and is represented by a single node. Composite change is an aggregation of atomic changes. A change contains metadata such as on which entity, by whom and when a change is requested

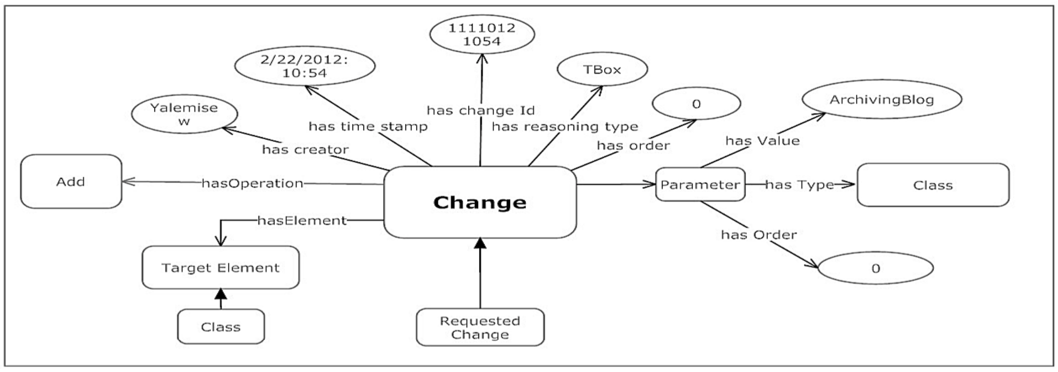


Figure 4.10: An example of atomic change operations

and implemented. It also contains the change operation, the OCMS element, the specific entity and other related information about the change. This information is treated as a node in the graph and is linked to the change node using edges with descriptive labels. A complete specification for an atomic change operation has the following information.

Operation. The action we want to implement in the ontology is represented by the operation. The operation can be addition, deletion or modification in case of atomic change operations and merge, copy, split, etc., in case of composite change operations. We represent modification as a series of additions and deletions, thus, the operation mainly contains addition and deletion operations. An addition operation introduces an entity which was not present in the OCMS in the previous version. A deletion operation removes an existing entity from the OCMS.

A change has an operation which can be either addition or deletion. The change and the operation nodes are connected by an edge represented as a directed arrow with a label *hasOperation*.

Target Entity. A target entity represents the changing entity of the OCMS. The type of the entity can be a class, an object property, data property, restriction, axiom, or instance which are defined in the OCMS graph. The target element is represented as a node and is connected to the change node with a directed edge *hasTargetEntity*. For example, a change operation which adds a class can be represented using three nodes, the change node, the operation node, which is *add*, and the target element node, which is *class*.

Parameter. A parameter represents one or more of the actual entities involved in the change, in our case the IRIs. A change may have one or more parameters. Each of the parameters has attributes to distinguish one from the other.

The parameter value attribute indicates the value of the specific parameter. For example, the above change operation can be applied to the parameter *cngl:#person*

The parameter order indicates the order in which the parameters appear in the change. The order indicates the dependent and antecedent entities. For example, a parameter with order equal to 0 indicates that the parameter appears at the beginning of the change operation.

The parameter type indicates the type of the parameter. In the above example, the type of the parameter is class, which indicates that this specific parameter is a class in the OCMS. This attribute gives important information when we have mixtures of parameters in the change. For example, Add classAssertion (*cngl:#inst1*, *cngl:#Person*) the first parameter with order 0 is *Inst1* and its type is instance. The second parameter with parameter order 1 is *Person* and its type is class. A parameter is connected to the change node using a directed edge labelled *hasParameter* and a change operation may have more than one parameter.

Creator. The creator represents the current user who requested the change operation. This node is essential to provide information about who requested and implemented the change operation. This node is connected to the change node using a directed edge labelled *hasCreator*.

Time stamp. A time stamp is used to record the time at which the change operation is implemented. This node stores the date and the time the change is implemented. It includes the seconds in microseconds. This node is connected to the change node using a directed edge *hasTimeStamp*.

Change Id. Every change needs to have a unique identifier to separate it from other changes in a change log. Change Id represents the value that is used to identify a given change uniquely. The change node is connected to the change Id node with a directed edge labelled *hasChangeId*.

Change order. When we represent a composite change operation, we want to keep the order at which an atomic change is executed. Change Order enables us to know which atomic change operation is executed first and which one follows next.

Statement type. The reasoning type represents the type of the statement the user is changing. The reasoning type is either $\mathcal{AB}$ox or $\mathcal{TB}$ox statement. This information serves as an input for the change impact analysis process.

A change can be a requested change or a derived change. Two special nodes are used from the change node to indicate the change is either a requested change or a derived change. ***A requested change*** is a change which is captured as an explicit change request. ***Derived changes*** are changes that are automatically generated to correctly implement the requested change in a given ontology. ***A complete change*** is a change which is the union of the requested change and the derived changes. Capturing this information is essential to determine the order of execution of the complete change operation. The following table explains the actual information stored about a change operation in an xml file.

```
<FinalChange>
     <Change>
        <ChangeId>1</ChangeId>
        <TimeStamp>2012/05/22 18:59:47:5947</TimeStamp>
        <ChangeType>generatedChange</ChangeType>
        <Creator>Yalemisew</Creator>
        <Order>1</Order>
        <ChangeOperation>Add</ChangeOperation>
        <TargetEntity>SubClassOf</TargetEntity>
        <Parameter Type="Class" Porder="0">
```

```
       <http://www.cngl.ie/University.owl#MastersStudent>
      </Parameter>
      <Parameter Type="Class" Porder="1">
       <http://www.cngl.ie/University.owl#Person>
      </Parameter>
      <StatementType>TBox</StatementType>
   </Change>
   <Change>
      <ChangeId>2</ChangeId>
      <TimeStamp>2012/05/22 18:59:47:5947</TimeStamp>
      <ChangeType>generatedChange</ChangeType>
      <Creator>Yalemisew</Creator>
      <Order>2</Order>
      <ChangeOperation>Add</ChangeOperation>
      <TargetEntity>SubClassOf</TargetEntity>
      <Parameter Type="Class" Porder="0">
       <http://www.cngl.ie/University.owl#PHDStudent>
      </Parameter>
      <Parameter Type="Class" Porder="1">
       <http://www.cngl.ie/University.owl#Person>
      </Parameter>
      <StatementType>TBox</StatementType>
   </Change>
   ...
</FinalChange>
```

4.5.3 Graph-based Formalization of Change Operations

The above change operations are applied to the OCMS graph (G) discussed in Section 4.4. The OCMS graph contains several nodes (N) and edges (E) which are subject to change. The representation of each node and edge is given in detail in Section 4.4. Thus, all the changes are applied either on the nodes or on the edges of the OCMS graph. For example, when we delete a content document, we are deleting a node from the content graph. When we remove a subClass axiom from the ontology graph, we are deleting a specific edge that links two nodes. We formally represent change operations using graphs. The formalization process begins from atomic change operations since composite and domain-specific changes are constructed from atomic change operations. To create composite or domain-specific changes, we need to combine atomic changes together. The target context of the operations (Section 4.4) is an OCMS graph $G = (N, E)$ where N is the set of nodes $N_1, N_2, \ldots, N_l$ and E is the set of edges $E_1, E_2, \ldots, E_m$ where $E_k = (N_i, \alpha_k, N_j)$ and $i, j \in \{1, 2, \ldots, l\}$ and $k \in \{1, 2, \ldots, m\}$.

4.5.3.1 Atomic Change Representation

Atomic change can be viewed as a change operation that adds or removes a single node or edge from the ontology.

- **Add Entity.** Given an OCMS graph $G = (N, E)$, an entity M and its node type T, the $Add\ Entity(M : T)$ operation results in a graph $G' = (N', E')$ where $N' = N \cup \{M\} \wedge E' = E \cup \{(M, \alpha, T)\}$ where $\alpha = rdf : type \wedge T \in \{owl : class, owl : dataProperty, owl : objectProperty, owl : instance\}$.

- **Add Axiom.** Given an OCMS graph $G = (N, E)$ and an axiom $A = \{(m_i, \alpha, m_j)\}$, the $Add\ Axiom(A)$ operation results in a graph $G' = (N, E')$ where $E' = E \cup A$.

- **Delete Entity.** Let $n \in N$ be the entity node to be deleted and $A = \{(n, \alpha, T)\} \in E$ be the axiom defining the type(T) of the entity, the $Delete\ Entity(n : T)$ operation on an OCMS graph $G = (N, E)$ results in a graph $G' = (N', E')$ where $N' = N - \{n\} \wedge E' = E - A$.

- **Delete Axiom.** Given an OCMS graph $G = (N, E)$ and an axiom to be deleted $A = \{(n_i, \alpha, n_j)\} \in E$, the $Delete\ Axiom(A)$ operation results in a graph $G' = (N, E')$

where $E' = E - A$.

The above formalization is very general and does not distinguish between specific implementations of change operations on specific entities. Thus, a detailed version of the formalization for specific entity types is given below. A detailed discussion of the description and the semantics of entities is given in Section 2.3.4.

- **Add Class.** Given an OCMS graph $G = (N, E)$ and a class node C, the Add $Class(C)$ operation results in a graph $G' = (N', E')$ where $N' = N \cup \{C\} \wedge E' = E \cup \{(C, rdf : type, owl : class)\}$.

- **Add Data Property.** Given an OCMS graph $G = (N, E)$ and a data property node DP, $Add\ DataProperty(DP)$ operation results in a graph $G' = (N', E')$ where $N' = N \cup \{DP\} \wedge E' = E \cup \{(DP, rdf : type, owl : dataProperty)\}$.

- **Add ObjectProperty.** Given an OCMS graph $G = (N, E)$ and an object property node OP, $Add\ ObjectProperty(OP)$ operation results in a graph $G' = (N', E')$ where $N' = N \cup OP \wedge E' = E \cup \{(OP, rdf : type, owl : objectProperty\}$.

- **Add Individual.** Given an OCMS graph $G = (N, E)$ and an individual node I, Add $Individual(I)$ operation results in a graph $G' = (N', E')$ where $N' = N \cup \{I\} \wedge E' = E \cup \{(C, rdf : type, owl : individual)\}$.

- **Delete Class.** Let $C \in N$ be the class node to be deleted and $A = \{(C, rdf : type, owl : class)\} \in E$ be the axiom defining the type of the node, the operation $Delete\ Class(C)$ applied on an OCMS graph $G = (N, E)$ results in a graph $G' = (N', E')$ where $N' = N - \{C\}$ and $E' = E - A$.

- **Delete DataProperty.** Let $DP \in N$ be the data property node to be deleted and $A = \{(DP, rdf : type, owl : dataProperty)\} \in E$ be the axiom defining the type of the node, then the operation $Delete\ DataProperty(DP)$ applied to an OCMS graph $G = (N, E)$ results in a graph $G' = (N', E')$ where $N' = N - \{DP\} \wedge E' = E - A$.

- **Delete ObjectProperty.** Let $OP \in N$ be the object property node to be deleted and $A = \{(OP, rdf : type, owl : objectProperty)\} \in E$ be the axiom defining the type of the node, then the operation $Delete\ ObjectProperty(OP)$ applied to an OCMS graph $G = (N, E)$ results in a graph $G' = (N', E')$ where $N' = N - \{OP\} \wedge E' = E - A$.

- **Delete Individual.** Let $I \in N$ be the individual node to be deleted and $A = \{(C, rdf : type, owl : individual)\} \in E$ be the axiom defining the type of the node, the operation $Delete\ Individual(I)$ applied to an OCMS graph $G = (N, E)$ results in a graph $G' = (N', E')$ where $N' = N - \{I\} \wedge E' = E - A$.

4.6 Evaluation

The layered operator framework is proposed to represent change requests and make them available for implementation. The evaluation focuses on the adequacy of the layered operator framework. Details of the evaluation are presented in the following subsections.

4.6.1 Adequacy of the Layered Operator Framework

The research problem in this section focuses on the representation of changes using change operations that are adequate for implementation and suitable for change impact analysis. Users of the OCMS require adequate change operations to represent changes and analyse impacts.

Adequacy measures whether the proposed operator framework is sufficient to represent the requested change. We evaluate the adequacy of the layered operator framework using an experiment.

4.6.1.1 Experimental Setup

We built a prototype which implements the proposed operator framework, as a proof of concept which includes a facility to specify the change request using operations supported in the layered operator framework. The prototype supports a total of 62 addition and deletion operations. This includes 14 change operations acting on classes, 12 operations acting on object properties, 12 operations acting on data properties, 12 operations related to individuals and individual assertions, 6 operations dealing with cardinalities, 4 operations dealing with restrictions and 2 change operations dealing with content. There are two operations dealing with document change and the remaining operations apply across entities of the ontology and the annotation. We further support 8 composite change operations [Stojanovic, 2004]. Any other evolving constructs supported by OWL 2.0, which are not covered in this experiment (such as annotation properties), are the limitations of this experiment.

To evaluate the adequacy of the proposed layered operator framework, we use changes derived from the Software Help File OCMS (Appendix A), which is built to semantically enrich software help files with domain ontologies, the Database Course OCMS (Appendix B), which is built to describe content in database systems course ware, and the University OCMS (Appendix C), which focuses on enriching a university administration system using semantics.

For the purpose of the experiment, we identified 10 change requests from empirically identified scenarios [Abgaz et al., 2011]. These change requests represent the majority of frequently observed operations and frequently evolving entities. To avoid the bias of a single user, we included additional users to participate in the implementation of the change operations. The first user is an expert in ontology evolution and the second user has know-how of ontologies and ontology applications, and the third user comes from the software engineering domain. Each user spends sufficient time to represent the individual change requests using the operators provided at each layer. The system implements the change requests and the users are asked to evaluate the adequacy of the change operations.

Before the experiment, we provided a short introduction of ontology constructs such as classes, data properties, object properties, etc., and how to specify change requests. We introduced the users to the overall environment of the prototype. During the experiment, we asked each user to implement the selected changes using the change operations provided by the system.

4.6.1.2 Experimental Results and Discussions

We evaluated the operator framework using all atomic change operations, selected composite and domain-specific change operations. We further collected feedback from users who participated in the evaluation of the prototype. We asked the users to implement selected change operations (Table 6.12) using the prototype and to evaluate the adequacy of the change operations to represent change requests. The results of the user evaluation are represented in Table 4.2.

Table 4.2: Adequacy of the layered operator framework

Rating	User 1	User 2	User 3
Fully Adequate	10	2	0
Adequate	0	8	3
Slightly Adequate	0	0	7
Slightly inadequate	0	0	0
Inadequate	0	0	0
Fully inadequate	0	0	0

The evaluation results show that all users agree on the adequacy of the layered operator framework. The users strongly agree on the adequacy of 40% of the change operations, agree on 36% of the change operation and 23% slightly agree on adequacy of the change operations. From the evaluation result, we found out that the layered operator framework provides adequate change operations to represent change requests. The advantage of the layered operator framework is that its composite and domain-specific change operations are composed from atomic change operations and the composition does not put any restriction on the number and order of atomic change operations. Thus, the layered operator framework meets its objective and is capable of representing the changes captured in an OCMS.

4.7 Summary

Two frameworks are proposed in this chapter. The first one focuses on defining the overall framework of an OCMS system and provides specifications on how each component is viewed in this research and how the components communicate each other. This framework paves the way for understanding the interaction and dependency between the entities in the layers. It systematically organizes the OCMS and makes it suitable for the proposed change impact analysis.

The proposed framework consists of the ontology layer at the top which may incorporate a single or multiple ontologies distributed over the web or stored on a single machine. The ontologies can be generic ontologies or domain specific. This layer provides semantics to the target content. The annotation layer uses triples to annotate content and content artefacts. The annotation layer uses triples to annotate content. The triple is organized as subject, predicate and object. It further incorporates context information to indicate the context of the triple. The content layer provides storage and retrieval of the content in the content-based system. Access to the content or part of the content is based on the document identifiers assigned to each content or part of the content. At this stage, this layer supports structured and semi-structured content.

The second framework focuses on defining the changes in OCMS and organizes them using four layers. This framework identifies the atomic change operations and their actions in the OCMS graph. It allows creation of composite and domain-specific change operations. This framework lays a foundation for change request representation, dependency analysis and change impact analysis.

The two frameworks clarify the OCMS and help us to understand the interaction between different entities within and across the layers. They further clarify the composition of changes from the atomic changes to abstract level changes. The frameworks support traceability of entities of each layer. The graph-based representation facilitates efficient searching and processing of entities and changes. The evaluation result of the operator framework shows that the framework is adequate to represent change requests. It is useful to represent composite and domain-specific change operations based on the preference of the user.

Chapter 5

Change Analysis Framework

5.1 Introduction

In the previous chapter, we gave an overview of the OCMS architecture and the layered operator framework. The framework identified the components of an OCMS, the evolving entities and the change operations. We represented the OCMS using graphs and formalized the change operations.

In this chapter, we present the change impact analysis framework. The overall change impact analysis framework includes the requested changes (Section 4.5.2.1), evolving entities, dependency analysis, evolution strategies and the core change impact analysis process. This chapter contributes the major inputs for the change impact analysis (Chapter 6) and for the change optimization and implementation phases (Chapter 7).

Together with the framework, in this chapter, we cover the dependency analysis and the evolution strategies. The dependency analysis identifies dependent entities that evolve together. The evolution strategy determines how a requested change operation is implemented. A single change request can be implemented using different evolution strategies. Each strategy composes changes differently and impacts the OCMS differently. Algorithms that are used to identify dependent entities, dependency rules and methods for combining dependency types with evolution strategies are the major focus.

This chapter is organized into six sections. Section 5.2 discusses the overall framework of change impact analysis. Section 5.3 discusses dependency analysis algorithms used to identify entities that are affected by a change operation. In Section 5.4, we present different change strategies, which affect the implementation of the requested change and the impacts of the requested change used during evolution. In this section, a customized implementation of strategies is presented. Section 5.5 provides the evaluation of the dependency analysis method and the overall change impact analysis framework. Finally we give a summary of the chapter in Section 5.6.

5.2 The Change Impact Analysis Framework

In this section, we give a brief introduction of the change impact analysis framework. The overall change impact analysis framework contains three major phases. The first phase receives change requests and represents them using change operations. This phase uses evolution strategies and dependency analysis to generate complete change operations. The second phase takes the represented changes and analyses the impacts of the change operations. This phase merges integrity analysis and change impact analysis together for efficient processing. Finally, we have the change implementation phase which allows the user to implement the changes based on the results of the impact analysis. Figure 5.1 outlines the phases of the change impact analysis framework and their interactions.

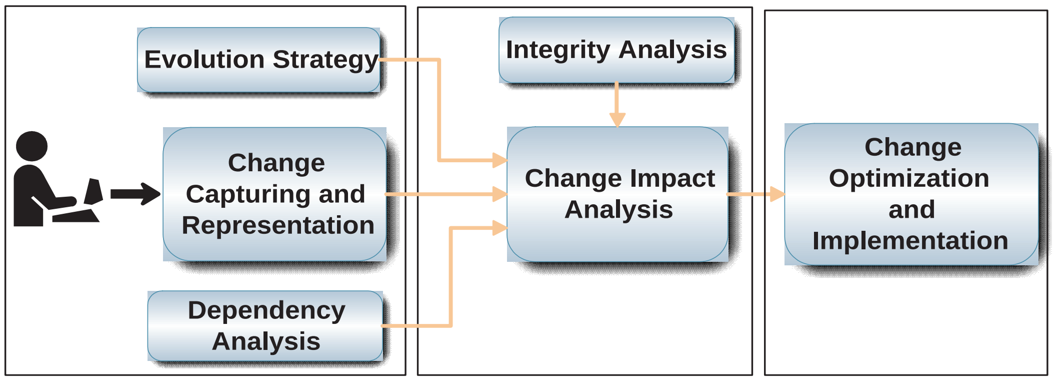


Figure 5.1: The change impact analysis framework

5.2.1 Change Request Capturing and Representation

The objective of this phase is to represent detected changes using suitable change operations (Section 4.5) that ensures the efficient implementation of the required change. The execution depends on how the change is represented and relies on two factors. The first factor is the selection of the appropriate change operator [Stojanovic, 2004]. The second factor is the order of execution of the operations focusing on efficient ordering of atomic change operations into composite and higher-level granularity to minimize impacts [Lee et al., 2000] [Arnold, 1996]. Change representation uses different evolution strategies and the output of the dependency analysis. The detailed discussion of the change request capturing and representation including dependency analysis and evolution strategy is presented in Section 5.3 and Section 5.4 respectively.

5.2.2 Change Impact Analysis

This step mainly focuses on determining the impacts of the captured change operations on the entities of the ontology. The impact determination process focuses on analysing the nature of the operations and the target ontology entities using different parameters. Based on these parameters, this phase categorizes change operations into different categories of impacts. The impact determination process is done using two phases. The first phase is individual change impact analysis. When a composite change operation is implemented, the impacts of the composite change may not be the same as the aggregation of the impacts of its constituent individual atomic change operations. Thus, the second phase is composite change impact analysis.

It further deals with the integrity of the overall system. The ***integrity*** of the OCMS focuses on the ***satisfiability*** of the ontology and the ***consistency*** of the annotation. In general satisfiability, checks whether a class expression does not necessarily denote the empty class and consistency refers to verifying whether every class in the ontology corresponds to at least one individual [Baader et al., 2003]. Consistency checks any contradiction of the facts in the annotation and shows the absence of contradiction focusing on individuals

[Stojanovic, 2004]. Using the change impact analysis results, we analyse the satisfiability of the ontology entities and the *consistency* of the annotation. Consistency is analysed based on consistency rules that are defined for the ontology.

Thus, we deal with the following widely used rules related to satisfiability of classes [Stojanovic, 2004].

- Identity invariant: no two entities should have the same id (URI).
- Rootedness invariant: there should be a single root in the ontology.
- Concept hierarchy invariant: no entity should have a cyclic graph.
- Closure invariant: every class should have at least one parent class except the root class.
- Cardinality invariant: the cardinality of a constraint should be a non-negative integer greater than or equal to the minimum cardinality and less than or equal to the maximum cardinality.
- User-defined constraints: these constraints are user-defined and need to be stated in the way they can be implemented like the other invariants.

Instances in OCMS are linked to the ontology using semantic annotation. Thus, determining the impact of change operations in relation to the instances is crucial. The determination of the $\mathcal{AB}$ox validity is based on consistency rules. These rules determine how instances/ instance properties should exist in the ontology structurally and how they should be interpreted:

- Invalid instance: given a consistent ontology, if there is an instance that does not correspond to any of the classes, then that instance is invalid.
- Invalid interpretation: given a consistent ontology, if there is an instance whose interpretation contradicts any interpretation denoted by the consistent ontology, that instance has an invalid interpretation.

For example, if the ontology specifies a student can not be both *MScStudent* and *PhDStudent* at the same time, but if we have *John* as an instance of *MScStudent* and *PhDStudent*, the instance is considered as invalid instance and it introduces invalid interpretation.

The change impact analysis process follows ex ante evaluation which begins during the change request stage of the evolution by collecting and analysing the change operations, the impacts and the causes of the impacts before the change is permanently implemented in the system. This reduces the effort required to roll back the changes if unwanted impacts are observed after a permanent implementation of the changes.

5.2.3 Change Optimization and Implementation

The change implementation phase takes the final change operations and executes them in the OCMS. This is done based on the user's preference after the impacts of the change operations are reviewed and approved by the user. This phase searches for optimal implementation using different optimization criteria such as severity of impacts, performance and type of statements changed. Change implementation is discussed in Chapter 7

5.3 Dependency Analysis for Change Representation

The change representation process takes the requested change and identifies the change operation (addition, deletion), the target entity (class, property, instance, restriction, and axiom) and the parameters. The requested change is represented using the graph discussed in the previous section. To implement the requested change, we need to process and determine if there are dependent entities that need to be changed in response to the requested change. The implementation of the requested change may vary depending on the target entity, the change operation and the evolution strategy (Section 5.4). Thus, the requested change operation alone may not be enough to evolve the OCMS and may require additional change operations. This forces us to conduct a dependency analysis to find dependent entities that change together with the target entity.

Understanding how the entities in the OCMS depend on each other is a crucial step for analysing how the change of one entity affects the other [Cox et al., 2001]. Characterization, representation and analysis of dependencies within and among the ontology, the annotation and the content layers are crucial aspects of change impact analysis. In this section, we present relevant dependencies which are identified from the structure of the OCMS system. These dependencies are useful for deriving additional changes to complete the requested change [Abgaz et al., 2012] [Abgaz et al., 2011]. All the dependencies that exist in the graph may not be important for dependency analysis. Thus, we identify the dependencies that are useful for implementing changes and analysing their impacts. We formally define such dependencies and present an algorithm to identify the dependent entities and their dependency types from a given ontology. This phase, together with the implementation strategies, forms an input for the change impact analysis process.

Dependency is defined as a reliance of one node on another node to get its structural and semantic meanings. For a node to be dependent on another node, it requires one or more edges that link it to the target node.

Given a graph $G = (N, E)$ and two nodes $N_1, N_2 \in N$, N_1 is dependent on N_2 represented by $Dep(N_1, N_2)$, if $\exists\ E_i \in E$ where $E_i = (N_1, \alpha, N_2)$. N_1 is the dependent entity and N_2 is the antecedent entity.

Dependency can be unidirectional or bidirectional. In the OCMS, we have edges that indicate bidirectional dependency ($A \leftrightarrow B$) called interdependence. Such interdependence is represented by axioms such as equivalence, disjoint, sameAs, differentfrom, etc. These kinds of dependencies can be mapped to two unidirectional dependencies $Dep(A, B)$ and $Dep(B, A)$.

Dependency analysis is the process of identifying the dependent entities, the dependency types and the characteristics of the dependencies of a given entity in the OCMS. The dependency analysis process takes an entity and the OCMS graph as an input, searches all dependent entities and returns a list of dependent entities for a given dependency. In OCMS, the three layers are interdependent. There are intradependence and interdependence among these components at a higher level. In such environments, we focus on the types of available dependencies, the formal representation of dependencies and the algorithms for identifying the dependencies. Analysing dependencies using different categories is beneficial in that most of the categories are useful for determining impacts when different change implementation strategies and configurations are used [Abgaz et al., 2010].

Structural Dependency Structural dependency refers to the hierarchical dependency or the taxonomic relationship between two nodes. When one node is dependent on the other node and if they are linked with edges that define the taxonomic relationships (subClassOf, subDataPropertyOf, subObjectPropertyOf, instanceOf), they become structurally dependent. These taxonomic relationships are expressed using subClassOf axioms between classes, subDataPropertyOf axioms between data properties, subObjectPropertyOf axiom between object properties and classAssertion axioms between instances and classes. Structural dependency also implies the semantic relationship between the entities.

Formally, for a graph $G = (N, E)$ and nodes $N_1, N_2 \in N$, N_1 is structurally dependent on N_2 is given by $strDep(N_1, N_2)$ if $\exists N_2.\ Dep(N_1, N_2) \wedge (N_1, \alpha, N_2)$ where $\alpha \in$\{subClassOf, subDataPropertyOf, subObjectPropertyOf, instanceOf\}.

5.3.1 General Properties of Dependency

Indirect Dependency. A dependency is said to be indirect, if there exist transitive or intermediate dependencies that link two nodes. Given a graph $G = (N, E)$ and nodes $N_1, N_2, N_3 \in N$, N_1 is ***indirectly dependent on*** N_3 represented as $indDep(N_1, N_3), if$ $\exists N_2.\ Dep(N_1, N_2) \wedge Dep(N_2, N_3) \wedge N_1 \neq N_2 \neq N_3$.

Total Dependency/ Partial Dependency. A total dependency refers a dependency when an entity is fully dependent on another entity for its existence. That means, there is no other dependency that enables it to get its meaning. A total dependency is observed when a target node depends only on a single node (articulation node).

Given a graph $G = (N, E)$ and nodes $N_1, N_2, N_3 \in N$, N_1 is ***totally dependent on*** N_2, represented by $TDep(N_1, N_2)$, if $\exists N_2.\ Dep(N_1, N_2) \wedge \neg\exists N_3.\ Dep(N_1, N_3) \wedge (N_2 \neq N_3)$.

A partial dependency refers to a dependency where the existence of a node depends on more than one node. Given a graph $G = (N, E)$ and nodes $N_1, N_2, N_3 \in G$, N_1 is ***partially dependent on*** N_2, represented by $Pdep(N_1, N_2)$, if $\exists N_2, N_3.\ Dep(N_1, N_2) \wedge Dep(N_1, N_3) \wedge (N_2 \neq N_3)$. Partial dependency is a complement of total dependency over all dependent entities. It is represented as:

$PDep = Dep - TDep$.

5.3.2 Types of Dependency

In this section we distinguish between different types of dependencies observed in an OCMS system. The dependencies are organized in relation to the OCMS layers.

We will use the OCMS snapshot in Figure 5.2 to elaborate the dependency analysis process throughout this chapter and the next chapter.

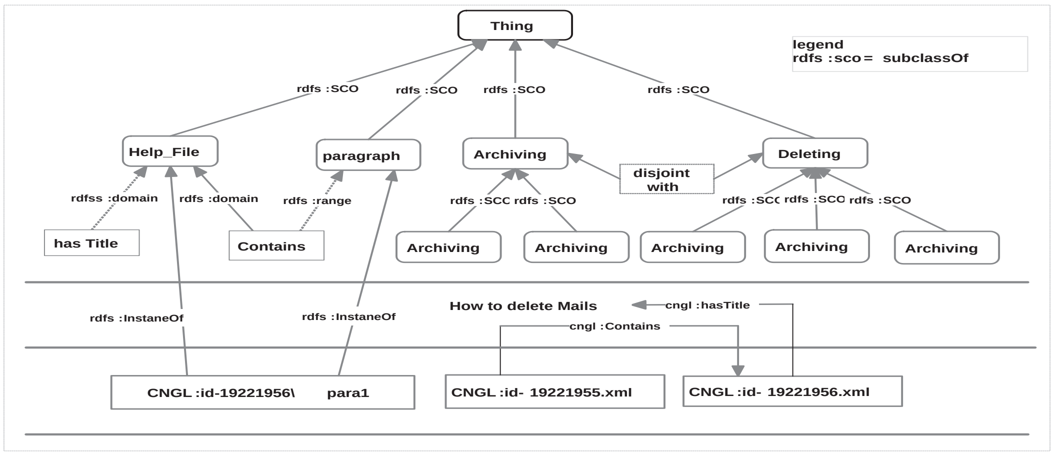


Figure 5.2: Software help OCMS - running example

5.3.2.1 Dependency within a Layer

Dependency in the Ontology Layer. The following dependencies between ontology entities are identified and their detailed definition is given below. The most frequent dependencies are presented here and the list can grow more when we represent complex class relationships. The context of the dependency is an OCMS graph $G = (N, E)$.

1. **Concept-Concept Dependency:** for a graph G and concept nodes $C_1, C_2 \in N$,

C_1 is ***dependent on*** C_2 represented by $CCDep(C_1, C_2)$, if $\exists C_2.\ Dep(C_1, C_2) \wedge (label(E_i = (C_1, \alpha, C_2)) = \text{“}subClassOf\text{”}) \wedge (type(C_1) = type(C_2) = \text{“}class\text{”})$. For example, there is a concept-concept dependency between *Activity* and *Archive*. *Archive* depends on *Activity* because there is an edge that links these two nodes with type *Class* and with node label *subclassOf*. Concept-concept dependency is transitive.

2. **Concept-Axiom Dependency:** for a graph G, a class node C_1, and any node $N_i \in N$ and an edge $E_i \in E$, E_i is ***dependent on*** C_1 represented by $CADep(E_i, C_1)$, if $(E_i = (C_1, \alpha, N_i) \vee E_i = (N_i, \alpha, C_1)) \wedge (type(C_1) = type(N_i) = \text{“}class\text{”})$. For example, if we take the concept “Activity”, there are three dependent *subClassOf* edges, one dependent *rdfs:range*. These axioms further characterize the dependency types.

3. **Concept-Restriction Dependency:** for a graph G, a class node C_1 and any node $N_i \in N$ and an edge $E_i \in E$, E_i is ***dependent on*** C_1 represented by $CRDep(E_i, C_1)$, if $E_i = (N_i, \alpha, C_1) \wedge (type(C_1) = \text{“}class\text{”} \wedge \alpha \in RA)$. For example, if we have a restriction *(isAbout, allValuesFrom, Activity)*, this specific restriction is dependent on the concept *Activity*.

4. **Property-Property Dependency:** for a graph G and a property nodes $P_1, P_2 \in N$, P_1 is ***dependent on*** P_2 represented by $PPDep(P_1, P_2)$ if $\exists P_2.\ Dep(P_1, P_2) \wedge (label(E_i = (P_1, \alpha, P_2)) = \text{“}subPropertyOf\text{”}) \wedge (type(P_1) = type(P_2) = \text{“}property\text{”})$. Here, property refers to both data property and object property.

5. **Property-Axiom Dependency:** for a graph G, a property node P_1, and any node $N_i \in N$ and an edge $E_i \in E$, E_i is ***dependent on*** P_1 represented by $PADep(E_i, P_1)$, if $E_i = (P_1, \alpha, N_i) \vee E_i = (N_i, \alpha, P_1) \wedge (type(P_1) = \text{“}property\text{”})$.

6. **Property-Restriction Dependency:** for a graph G, a property node $P_1 \in N$ and a restriction edge $R_1 \in E$, R_1 is ***dependent on*** P_1 represented by $PRDep(R_1, P_1)$ if $E_i = (N_1, \alpha, P_1) \vee E_i = (P_1, \alpha, N_1) \wedge (type(P_1) = \text{“}property\text{”})$.

7. **Axiom-Concept Dependency:** Given an axiom edge E_i and a concept node $C_1 \in G, C_1$ is ***dependent on*** E_i represented by $ACDep(C_1, E_i)$, if $E_i = (C_1, \alpha, N_i) \wedge (label(E_i) = \text{“}subClassOf\text{”}) \wedge (type(N_1) = \text{“}class\text{”})$. This dependency type is used to catch orphan concepts. If orphan concepts are not allowed in the ontology, we use such dependencies to find them.

5.3.2.2 Dependency across Layers

Content-annotation dependency. Content-annotation dependency refers to the dependency of the annotation on the actual content in the content layer. The content-annotation dependency occurs due to the fact that the annotation layer links the instances (documents) in the content layer with the ontology layer for attaching semantics to the content. If the content is changing, the dependent entities in the annotation layer (in this case those triples linked to the content) will be affected.

An annotation $A_i \in G_a$ is ***dependent on*** $d_i \in Cont$, represented by $AnCoDep(A_i, d_i)$, if exists $E_a = \{N_{ai}, \alpha_a, N_{aj}\} \in G_a$ such that $(N_{ai} = d_i) \vee (N_{aj} = d_i)$. This means A_i is dependent on document d_i if the document is used as a subject or an object of the annotation triple.

For example, the content document *CNGL:id-19221955.xml* is annotated as an *instanceOf* a $Help\_File$ and its type (*rdf:type*) is defined as *Instance* in the annotation layer. If the subject (*CNGL:id-19221955.xml*) of this annotation is removed all its dependent entities (*rdf:type*, *instanceOf*) will be affected.

Ontology-annotation dependency. Ontology-annotation dependency refers to the dependency of the annotation on the entities in the ontology layer. The annotation layer provides semantics for the content using entities from the ontology. Whenever a change is made to an entity in the ontology, all the dependent entities in the annotation layer will be affected. Generally, this dependency is represented as follows.

An annotation $A_i \in G_a$ is ***dependent on*** $O_i \in G_o$ represented by $AnOnDep(A_i, o_i)$, if exists $E_a = \{N_{ai}, \alpha_a, N_{aj}\} \in G_a$ such that $(\alpha_a = o_i) \vee (N_{aj} = o_i)$. This dependency across the layers is represented by three different dependencies in the OCM graph G as follows.

1. **Concept-Instance Dependency:** for a graph G and an instance node I_1 and a concept node $C_1 \in N$, I_1 is ***dependent on*** C_1 represented by $CIDep(I_1, C_1)$ if $\exists$ $E_i \in E$ where $E_i = (I_1, \alpha, C_1) \wedge (label(E_i) = \text{“}classAssertion\text{”}) \wedge (type(I_1) = \text{“}individual\text{”}) \wedge (type(C_1) = \text{“}class\text{”})$. For example, if we remove the class $Help\_file$, the dependent triples {(CNGL:id-19221955.xml, $instanceOf$, $Help\_file$) and *(CNGL:id-19221956.xml, instanceOf,* $Help\_file$)} will be affected. This indicates that those annotations are dependent on the concept in the ontology layer.

2. **Property-Instance property Dependency:** for a graph G and an instance property node IP_1, and any node N_i, N_j and a property node $P_1 \in N$, IP_1 is ***dependent on*** P_1 represented by $PIPDep(IP_1, P_1)$ if $\exists$ $E_i \in E$ where $E_i = (N_i, \alpha, N_j)$ such that $(label(E_i) = P_1) \wedge (type(N_i) = \text{“}instance\text{”}) \vee (type(N_j) = \text{“}instance\text{”})$. For example, in ($CNGL : id19221956.xml$, *cngl:hasTitle*, *“How to delete Mails”*) the instance property *cngl:hasTitle* is dependent on the property *hasTitle* in the ontology layer.

3. **Instance-Axiom Dependency:** for a graph G, an instance node I_1, and any node $N_i \in N$ and an edge $E_i \in E$, E_i is ***dependent on*** I_1 represented by $IADep(E_i, I_1)$, if $(E_i = (I_1, \alpha, N_i) \vee E_i = (N_i, \alpha, I_1)) \wedge (type(I_1) = type(N_i) = \text{“}instance\text{”})$.

4. **Axiom-Instance Dependency:** for a graph G and an instance node I_1 and an edge $E_i \in E$, I_1 is ***dependent on*** E_i represented by $AIDep(I_1, E_i)$ if $E_i = (I_1, \alpha, N_2) \wedge (label(E_i) = \text{“}instanceOf\text{”}) \wedge (type(i_1) = \text{“}instance\text{”})$.

All edges that are linked to a node or all nodes that are linked together do not necessarily show dependency. For example, an instance property is dependent on the definition of the corresponding property. However, a property is not dependent on its instance properties. The focus of this research is on identifying and formalizing dependencies that represent propagation of impacts in the OCMS. Using these dependencies, we developed algorithms to identify dependent entities.

5.3.3 Dependency Analysis Algorithm

It is important to formally identify the dependencies and their types to determine the impacts of a change operation. However, manually identifying these dependent entities and the type of the dependency is difficult. Thus, we developed an algorithm which identifies dependent entities and the dependency types. The algorithm starts from the universal entity “Thing” and filters out the dependent nodes based on the dependency definitions. We customized the general tree search algorithm [Heckel, 2006] to identify dependent entities. The search algorithm checks the edges and the nodes which are linked to the target node and matches it with the defined dependencies. We move to the dependent nodes and repeat the search by examining the types of the nodes. We end the search when there is no more node and edge to be visited. The individual dependency analysis algorithms are discussed as follows.

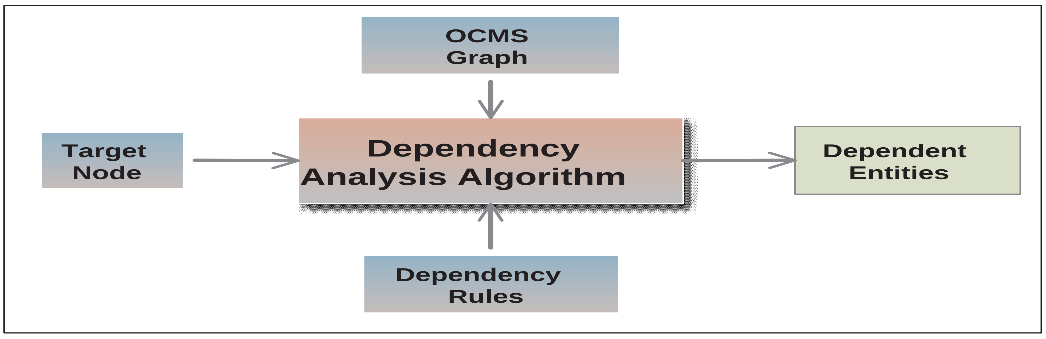


Figure 5.3: Dependency analysis diagram

5.3.3.1 Direct Dependent Entities

Getting direct dependent entities of a target entity is done by examining each of the edges that point to the target entity. For example, if the target entity is a class node, and if there is an edge with label *subClassOf* (represented as SC in Figure 5.4) that links other class nodes to this target entity, and then based on concept-concept dependency, we can identify the dependent entities. If there is a single edge with label *subClassOf* between the two edges, then we consider those dependent instances as direct dependent entities. In Figure 5.4 all the direct dependent concepts of the concept *Activity* are highlighted in gray and the direct dependent axioms are represented by dotted lines.

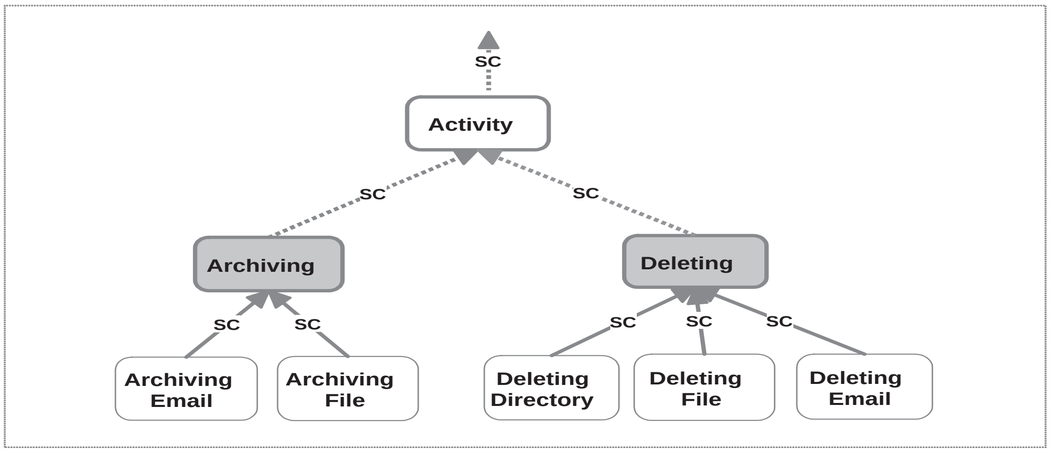


Figure 5.4: Direct dependent classes

Not all nodes that are linked to the target entity may indicate dependency between two nodes. Here, we need to examine the edges that connect other nodes to the target entity. A node is considered as a dependent node only when it satisfies one or more of the dependencies defined in Section 5.3.2. The algorithm for identifying direct dependent concepts is given below (Algorithm 1). This algorithm is tuned to identify direct dependent classes. The identification of dependent instances, axioms and restrictions is done in a similar way by customizing the parameters and the dependency rules.

Algorithm 1 getDirectDependentClasses(G,c)

```
1: **Input:** Graph $G$, Class node $c$
2: **Output:** direct dependent classes ($d$)
3: $d \leftarrow \emptyset$
4: **if** the node $c$ exists in $G$ **then**
5:     **for** each edge $E_i = (m, \alpha, c)$ directed to $c$ **do**
6:         **if** $label(E_i) = “subClassOf” \wedge type(m) = “class”$ **then**
7:             add $m$ to $d$
8:         **end if**
9:     **end for**
10: **end if**
11: return d
```

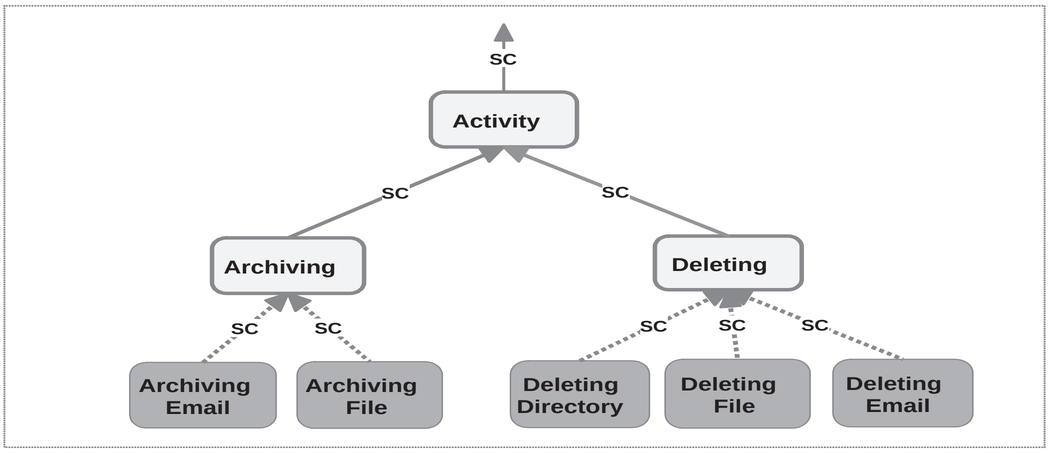


Figure 5.5: Indirect dependent classes

5.3.3.2 Indirect/transitive Dependent Entities

Identifying indirect dependent entities of the target entity is done in the same way as the method used for direct dependent entities. The main extension for identifying transitively dependent entities is taking all direct dependent entities as a target entity and to run the direct dependency on them recursively until we reach a leaf node - the node that does not have a dependent node.

For example, to get all the direct and transitive dependent entities of class *Activity*, first, we call get direct dependent entities (*Archiving* and *Deleting*), then taking them as an input, we further get direct dependent entities of (*Archiving*) which are *ArchivingEmail* and *ArchivingFile*. Then, we further move to *Deleting* and we get *DeletingDirectory*, *DeletingFile* and *DeletingEmail*. In Figure 5.5, the nodes highlighted in gray are indirect dependent classes of *Activity* and the edges represented by broken lines are the indirect dependent axioms.

We stop the process when we reach a node that contains no more dependent nodes. To prevent infinite recursion, we store information about the previously expanded nodes and check if we already visited the nodes earlier. Here, we are interested in both direct and transitive dependent entities. In this case we expand the above algorithm to include the transitive dependent entities.

Algorithm 2 getAllDependentClasses(G,c)

```
1: Input : Graph $G$, Class node $c$
2: Output: all dependent classes=$d$
3: $d \leftarrow \emptyset$
4: Queue $Q$
5: if the node $c$ exists in $G$ then
6:     DirectDep $\leftarrow$ getDirectDependentClasses(G,c)
7:     for each concept $c_i$ in DirectDep do
8:         $Q.push(c_i)$
9:         if $c_i$ not in $d$ then
10:            add $c_i$ to $d$
11:        end if
12:    end for
13:    while $Q$ is not empty do
14:        $Temp = Q.peek()$
15:        getAllDependentClasses(G,Temp)
16:        $Q.remove()$
17:    end while
18: end if
19: return d
```

We used the breadth-first search strategy [Cormen et al., 2001] [Heckel, 2006] to traverse through the OCMS graph and identify the dependent entities using dependency types defined in Section 5.3.2. Algorithm 2 requires the whole OCMS graph and the target node or edge as an input and returns nodes and edges that are dependent on the target entity. A breadth-first algorithm fits our requirements for two reasons. First, the algorithm guarantees identification of all dependent entities that exist in the graph. Second, it allows us to preserve the vertical hierarchies of the graph. We get all the directly dependent classes of a class first rather than moving down to the indirect subclass at the next level. All the subclasses at a given distance from the target entity can be identified using breadth-first search. We use the information to maintain the order of change operations. For example, in the case of deletion, we start deleting the class from the bottom nodes and advance up in the hierarchy until we reach at the target node.

5.3.3.3 Total Dependent Entities

To find all totally dependent entities of a given entity, we need to get all the dependent entities using Algorithm 2 which returns all dependent entities. Meanwhile, we check the dependent entities whether they are totally dependent on the given entity or not. Finding the totally dependent entities of classes is done by checking whether a given directly dependent class has more than one super class or not. If the directly dependent class has more than one super class, it becomes a partially dependent class. However, this approach does not guarantee us totally dependent entities when it is extended to transitively dependent classes. A class can have more than one super class but can still be totally dependent on a given class. A direct dependent class is totally dependent, if it has only one super class. But, indirectly dependent classes may not satisfy this rule. A recursive implementation of the algorithm requires further customization to identify the total dependent entities.

Thus, we identified additional conditions to be checked to filter out the total dependent entities. If an indirectly dependent class has only one super class, or if all its super classes are totally dependent classes of the given class, it is considered to be a totally dependent class. However, if an indirect dependent class has a super class which is not in the dependent class list of the given class, then we consider that class as a partially dependent class. Algorithm 3 returns all totally dependent entities. In addition to that, it keeps track of the partially dependent entities. By changing the return value of the function to *partialDepCls*, we can also get all the partially dependent classes.

In Figure 5.6, all the highlighted class nodes are dependent class nodes on the class *Activity*. However, only *Deleting*, *DeletingDirectory* and *DeletingFile* are totally dependent classes. All the other class nodes (*Archiving*, *Archiving Email*, *ArchivingFile* and *DeletingEmail*) are partially dependent classes of the activity class. One interesting relationship we can see here is that the totally dependent classes (*ArchivingEmail*, *ArchivingFile*) of a partially dependent class (*Archiving*) are excluded from the totally dependent class list due to their parent class *Task*. The class *Task* is not dependent on *Activity*. This makes the *Archiving* class and its subclasses to be partially dependent on *Activity* class. Algorithm 3 returns

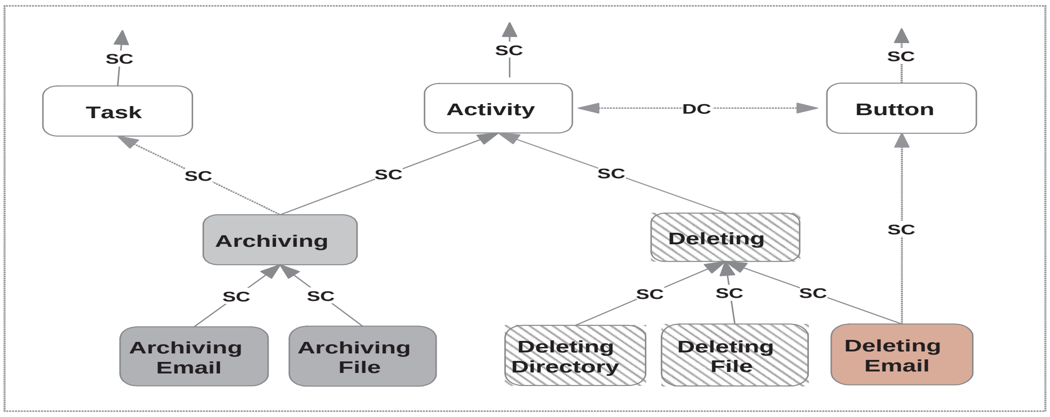


Figure 5.6: Total dependent and partial dependent classes

all class nodes that are totally dependent on the given class c.

5.3.3.4 Direct Total Dependent Entities

Direct total dependent entities are entities that are directly dependent and totally dependent on a given entity. Direct total dependent entities are entities that are the result of the intersection between total dependent and direct dependent entities. The intersection of the results of Algorithm 1 and Algorithm 3 gives us the direct total dependent entities.

5.3.3.5 Direct Partial Dependent Entities

Direct partial dependent entities are entities that are directly dependent and partially dependent entities. These entities play a major role in the impact analysis process and identifying them is crucial. However, they can be easily extracted once the direct and total dependent entities are identified. This means that direct partial dependent classes are the intersection of direct dependent classes and partial dependent classes. Figure 5.7 shows the properties of dependencies and their relationships. Any dependent entity, which is not totally dependent, is partially dependent entity.

Algorithm 3 getTotalDependentClass(G,c)

```
1:  **Input :** Graph $G$, Class node $c$
2:  **Output:** all total dependent classes=$d$
3:  $d \leftarrow \emptyset$, contained=true
4:  Set depCls=$\emptyset$ ,totalDepCls=$\emptyset$ ,partialDepCls=$\emptyset$, super=$\emptyset$
5:  depCls$\leftarrow$ getAllDependentClasses(G,c)
6:  **for** each concept $c_i$ in depCls **do**
7:      **if** count(getSuperClasses($c_i$)=1 **then**
8:          super $\leftarrow$ getSuperClasses($G, c_i$)
9:          **if** super not in partialDepcls **then**
10:             add $c_i$ to totalDepCls
11:         **end if**
12:     **else**
13:         super $\leftarrow$ getSuperClasses($G, c_i$)
14:         contained=true
15:         **for** each $sc$ in super **do**
16:             **if** $sc$ not in depCls **then**
17:                 contained=false
18:             **end if**
19:         **end for**
20:     **end if**
21:     **if** contained=true **then**
22:         add $c_i$ to totalDepCls
23:     **else**
24:         add $c_i$ to partialDepCls
25:     **end if**
26: **end for**
27: return totalDepCls
```

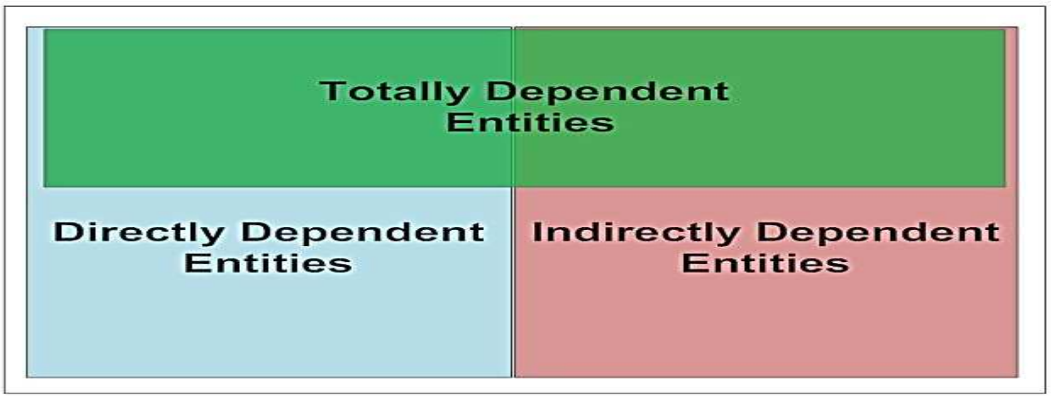

Totally Dependent Entities
Directly Dependent Entities
Indirectly Dependent Entities

Figure 5.7: Dependency diagram

Algorithm 4 getSuperClasses(G,c)

```
1: Input : Graph $G$, Class node $c$
2: Output: superclasses of a class
3: Set supCls←$\emptyset$
4: for each edge($C$, $\alpha$, $c_i$)in $G$ do
5:    if $\alpha =$ “$subClassOf$” and $type(c_i) =$ “$class$” then
6:       add $c_i$ to supCls
7:    end if
8: end for
9: return supCls
```

5.3.3.6 Combining all together

Following a similar procedure, we conducted a dependency analysis for properties, axioms, restrictions and instances. A complete dependency analysis of concepts combines individual dependencies and generates all the dependent entities. To identify that kind of dependency we call those individual algorithms.

Algorithm 5 getTotalDependentEntities(G,E)

```
1: Input : Graph $G$, an entity $E$
2: Output: Total Dependent entities
3: Set totalDepEnt←$\emptyset$ , totalDepCls←$\emptyset$
4: Set totalDepProp←$\emptyset$, totalDepInst←$\emptyset$
5: Set totalDepAxms←$\emptyset$
6: totalDepCls ← totalDependentClasses(G,E)
7: totalDepProp ← totalDependentProperties(G,E)
8: totalDepInst ← totalDependentInstances(G,E)
9: totalDepAxms ← totalDependentAxioms(G,E)
10: totalDepEnt ← totalDepCls $\cup$ totalDepProp $\cup$ totalDepInst $\cup$ totalDepAxms
11: return totalDepEnt
```

This algorithm accepts a target entity, which can be a class, a property, an instance, an axiom or a restriction and returns all the dependent entities. This algorithm simply calls previous algorithms to do the task and return the results. Then, it collects the results together and makes them ready for change representation based on the change implementations strategy selected. The difficulty associated with implementing this algorithm is that when we have bidirectional dependencies, the algorithm may exhibit a nonterminating behaviour. To avoid these cycles we store the already visited nodes. We check existing members in the queue to avoid duplication of entities. This increases the efficiency of the algorithm.

Limitation of the Dependency Analysis Algorithm There are some limitations involved in the analysis of total and partial dependency between classes and axioms representing complex classes. If the dependent entities contain complex axioms or complex class representations, the dependency analysis process involves complex decisions other than the issues addressed above. When an entity is deleted, complex axioms that are dependent on the entity may contain other entities. The relationship between the entities depends on how the complex axiom is formulated (conjunction or disjunction). To decide the dependency, we need to determine two things. First, using the dependency analysis we discussed above, we have to determine whether the axiom is totally dependent or partially dependent on the deleted entity. The second one and the more complex one is to determine whether there are other entities that are not yet captured but which are dependent on the axiom (axiom-concept dependency). A closer look at the following class expression explains the difficulty involved in determining the dependent entities.

```
equivalentClasses (http://cngl.ie/ EV-triples.owl#AdvancedUser
objectIntersectionOf (http://cngl.ie/ EV-triples.owl#User>
objectSomeValuesFrom(http://cngl.ie/ EV-triples.owl##hasExpertise>
http://cngl.ie/ EV-triples.owl#AdvancedExperience>)) )
```

This axiom states that *AdvancedUser* is a user who has advanced Experience. Assume that there is a change request that deletes *AdvancedExperience*, but the above axiom is using the concept in the property *hasExpertise*. If we delete *AdvancedExperience*, this axiom will be in the list of affected entities. If we do not conduct a further analysis, we will miss other entities that are involved in the axiom. For example, the deletion of this class makes the property very general because there is no domain defined for it. This does not distinguish *AdvancedUser* from other users semantically. The subtlety lies in how to determine such kinds of chained dependencies and identify the entities that are affected and how they are affected. Such kinds of complex axioms and representations require further and detailed investigation by looking at each candidate axiom and the extent of the change to other entities involved in the axiom.

5.4 Evolution Strategies

There are different ways of implementing a requested change in an OCMS. This depends on the selected change implementation strategy. These change implementation strategies are used to specify how a given change request is implemented. The change strategies determine how to fill the gap between the requested change and the changes required to correctly implement the user request. This includes consequential changes, which are not specified in the change request and corrective changes which are introduced to avoid inconsistencies. The different change implementation strategies are further used to avoid known violations of ontology rules (Section 6.3.1) and some of them are provided as change implementation strategies in existing ontology editors such as KAON [Volz et al., 2003] [Stojanovic, 2004] and Protege[1] [Knublauch et al., 2004]. The user can choose or set which strategy to follow before or at the time of the change implementation.

We identified four different strategies used by existing systems [Stojanovic, 2004] and customized them to provide additional implementation options for the users. These strategies are *no-action* strategy, *cascade* strategy, *attach-to-parent* strategy, and *attach-to-root* strategy. We will focus on the first three change implementation strategies. The attach-to-root strategy uses a similar technique as the attach-to-parent strategy. The only difference between the two is that the attach-to-parent strategy uses the immediate parent entity and the attach-to-root strategy uses the root entity (the top entity). We customize the attach and cascade strategies to be applied to both $\mathcal{TB}$ox and $\mathcal{AB}$ox statements or only to $\mathcal{TB}$ox statements. We further customize the strategies to N-level cascading (Section 5.4.4) to make the evolution process flexible. The details of each of the techniques used by the change implementation strategies are discussed below.

5.4.1 No-action Strategy

The no-action strategy states that a given change operation is implemented using the requested change without adding consequential or corrective changes. The final change oper-

[1]http://protege.stanford.edu/

ation does not include any other change operation than the ones that remove the references of the target entity from the OCMS. For example, when the user requests to add a class, it will be implemented as it is, without adding a subclass axiom to link it to a parent class. Another example is, when the user wants to delete a class, and if that class has subclasses, this strategy does not consider the subclasses of the target class but simply removes the target class and all the edges of the class. In our running example this means, when the user requests to delete the concept *Archiving*, we simply delete the concept by implementing the requested atomic change, delete Concept (*Archiving*) and all the axioms that refer to *Archiving* (Table 5.1).

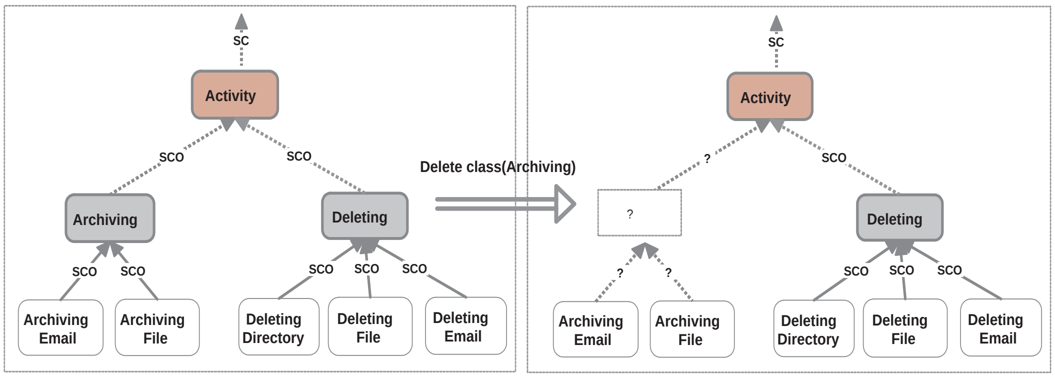


Figure 5.8: No-action strategy

Given the graph $G = (N, E)$ and a entity node $n \in N$, the *NO-ACTION* strategy is defined as follows:

$$\textit{NO-ACTION}\ (Delete\ Entity(n)) := \{Delete\ Entity(n), Delete\ Axiom(A) \mid \\ A \in directDependentAxioms(G, n)\}$$

Using the above strategy, we generate a change operation which deletes the class as follows.

5.4.2 Cascade Strategy

The cascade strategy states that whenever a change is requested, the change propagates to all dependent entities of the target entity. In OCMS, this means when we change some

Table 5.1: Generated changes - No-action strategy

Delete subClassOf(ArchivingFile, Archiving)
Delete subClassOf(ArchivingEmail, Archiving)
Delete subClassOf(Archiving, Activity)
Delete class(Archiving)

entity in the content-based system, we need to change all its dependent entities. In case of deletion, when we delete an entity, the deletion propagates to all its dependent entities. In case of addition, when we add an entity, we need to add all other entities that make the new entity semantically and structurally meaningful. Thus, whenever we use the cascade strategy, we identify the dependent entities of the target entity and we further introduce change operations that remove all these dependent entities. Thus, in cascade strategy, we generate intermediate change operations to implement the requested change in a cascaded mode to all dependent entities of the target entity (Table 5.2). In this delete concept (*Archiving*) results the deletion of all its subclasses and axioms.

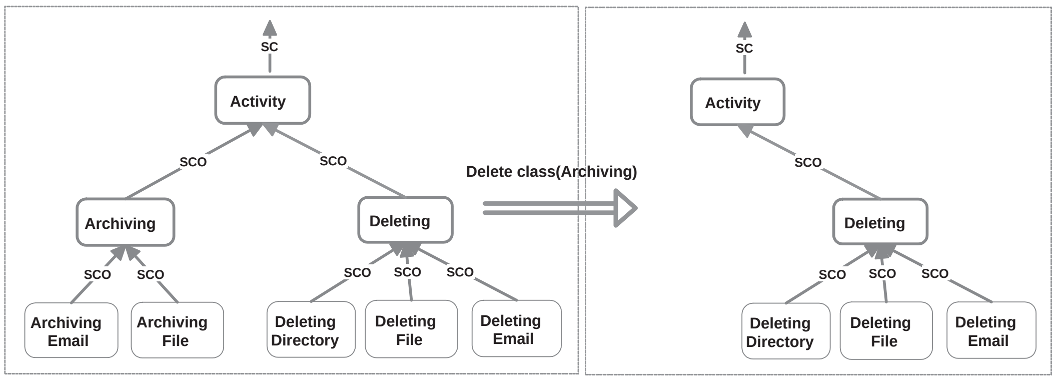


Figure 5.9: Cascade strategy

Given the graph $G = (N, E)$ and a entity node $n \in N$, the *CASCADE* strategy is defined as follows:

$$\textit{CASCADE}\ (Delete\ Entity(n)) := \{Delete\ Entity(n') \mid n' = n \lor \\ n' \in allTotalDependentEntities(G, n)\}$$

Using the above strategy we generate a change operation which deletes the class.

Table 5.2: Generated changes -Cascade Strategy

Delete subClassOf(ArchivingFile, Archiving)
Delete subClassOf(ArchivingEmail, Archiving)
Delete subClassOf(Archiving, Activity)
Delete class(ArchivingEmail)
Delete class(ArchivingFile)
Delete class(Archiving)

The cascade strategy can be further implemented in two ways. The first is cascading the change only to the dependent classes leaving the instances of the class, and the second one is cascading the change to the dependent classes and the instances. In the former case, the change is applied to the classes without changing the instances. This may cause the instances to lose some semantic meaning due to the removal of concepts which were providing additional meaning to the instances. It may even cause the instances to become orphan instances.

5.4.3 Attach-to-Parent/Root Strategy

The attach-to-parent strategy, or attach strategy in short, states that when a change is requested, link all the affected entities to the parent entity of the target class whenever it applies. This means, when a certain entity is deleted, link its dependent entities to the parent of the target entity whenever it applies. Thus, in the attach-to-parent strategy, we generate intermediate change operations in addition to the requested changes operations.

Given the graph $G = (N, E)$ and an entity node $n \in N$, the ATTACH strategy is defined as follows:

$$
\begin{aligned}
&ATTACH(Delete\ Entity(n)) := \{A, B, C \mid \\
&\qquad A := Add\ Axiom(A', n') \mid A' \in directDependentAxioms(G, n) \land \\
&\qquad\quad n' \in superEntity(n) \land \\
&\qquad B := Delete\ Axiom(A', n) \mid A' \in directDependentAxioms(G, n) \land \\
&\qquad C := Delete\ Entity(n)\ \}
\end{aligned}
$$

In this strategy for example, Delete class (*Archiving*) causes the deletion of the class and causes all the subclasses of the *Archiving* class to reconnect to the parent (*Activity*) class. Moreover, the class (*Archiving*) and all its related axioms will be deleted. Following the attach-to-parent strategy, we generate the necessary change operations in Table 5.3.

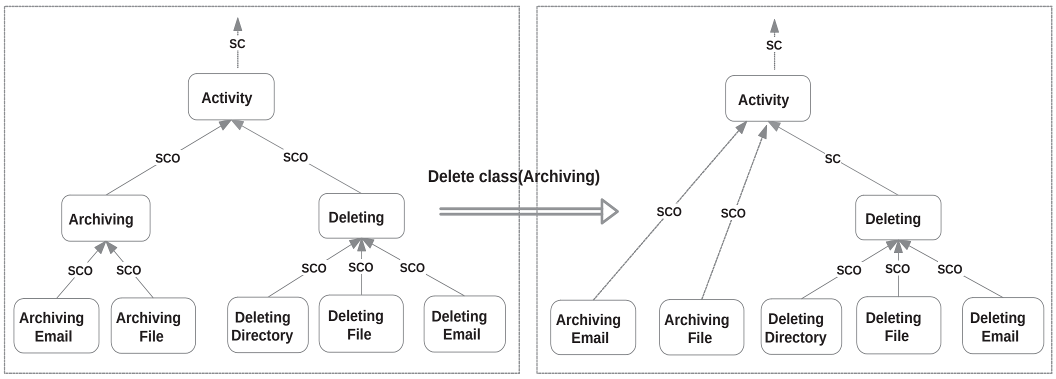


Figure 5.10: Attach-to-parent strategy

Table 5.3: Generated changes - Attach strategy

Add subClassOf(ArchivingFile,Activity)
Add subClassOf(ArchivingEmail,Activity)
Delete subClassOf(ArchivingFile, Archiving)
Delete subClassOf(ArchivingEmail, Archiving)
Delete subClassOf(Archiving, Activity)

There might be entities in the OCMS that are not compliant with the last two strategies. In such a situation, we further analyse the dependent entities and choose to implement one of the other strategies. However, all requested changes are implemented using at least one strategy.

The change implementation strategy is dependent on a specific OCMS. The content manager or the ontology engineer sets the change implementation strategy or the user is asked to confirm every time she/he requests a change. The change implementation strategy tells the system what action to take when the requested change affects entities other than itself. In this research, we deal with three change implementation strategies: NO-ACTION (N), CASCADE(C) and ATTACH (A). Two variants of the CASCADE and ATTACH strategies further distinguish between $\mathcal{AB}$ox and $\mathcal{TB}$ox statements. These change implementation strategies are used for generating complete change operations. For example, when the operation is delete class (DC) and the change implementation strategy is cascade-delete (C), we follow the DCC route and if the strategy is attach-to-parent, we follow DCA.

5.4.4 N-Level Cascading

N-level cascading is a customized form of the cascade strategy. This strategy cascades the change up to nodes that are found N distances from the target node. For example, if N is set to be 2, the N-level cascading will cascade the change up to two hierarchies. In Figure 5.10, when N= 1, and a deletion of the class *Activity* is requested, the two classes *Archiving* and *Deleting* will be removed with the target class. If N=2, all the subclasses down to two levels of hierarchy will be deleted, which includes *ArchivingEmail*, *ArchivingFile*, *DeletingDirectory*, *DeletingFile* and *DeletingEmail*. When we use N-level cascading, we may require using the attach-to-parent strategy to resolve orphan classes. In the above example, when N=1, once we delete the classes the subclasses of those deleted classes become orphans. To avoid the orphan classes, we use attach-to-parent strategy to link them with the parent of *Activity* class.

There are different strategies for composite change operations. Providing detailed discussion of the strategies in composite change operations is not the scope of this research. However, for further details, we direct the reader to [Javed et al., 2012].

5.4.5 Combining Dependencies and Strategies

A complete change operation is generated by taking the requested change, the dependency analysis and the selected evolution strategy. Determining the dependent entities is a crucial step that enables us to decide the extent of the effect of the requested change operation on the remaining entities of the ontology. For each of the cases, we look at the process of finding affected entities using dependency analysis. After conducting different experiments using our case studies (Appendix A to C), we determined the dependency types that can be associated with the change implementation strategies. In the case of no-action, we implement the requested change by analysing direct dependent entities; in the case of cascading, we are interested only in total dependent classes.

Deleting all dependent entities without checking whether they are totally or partially dependent entities causes a loss of entities that could still exist without violating the syntax or the semantics of the ontology. Thus, when we use the cascade-delete strategy, all total dependent entities, whether direct or indirect, will be deleted. In the case of the attach-to-parent strategy, our experiment showed that there are cases where we should consider partially dependent entities in addition to the total dependent entities. However, we only need the direct dependent entities.

Table 5.4: Combination of dependency with evolution strategy

Evolution Strategy	Total Dependency	Partial Dependency	Direct Dependency	Indirect Dependency
No-action	Ignore	Ignore	Apply	Ignore
Cascade	Apply	Ignore	Apply	Apply
Attach-to-parent	Apply	Apply (with exception)	Apply	Ignore

Table 5.4 shows the dependencies used in different strategies. In the case of the no-action strategy we use direct dependent entities. In the case of the cascade strategy, all direct and indirect, but only total dependent entities are used. Partially dependent entities are not affected. In the case of the attach-to-parent, we use all the direct dependent entities and attach them to the parent of the target entity. In this case, we use both total dependent entities and partial dependent entities with few exceptions. The exception applies for partial entities, which are already linked to the parent entity of the target entity.

A complete change operation contains the requested change operations and the generated change operations. Based on the type of the change request, individual changes are ordered accordingly to form a complete change operation. An example of a complete change operation in case of cascade-delete is given in Table 5.5.

Table 5.5: Complete change operations

Delete subClassOf(ArchivingFile, Archiving)
Delete subClassOf(ArchivingEmail, Archiving)
Delete subClassOf(Archiving, Activity)
Delete subClassOf(Archiving, Activity)
Delete class(ArchivingEmail)
Delete class(Archiving)

5.5 Evaluation

In this chapter, we evaluate the proposed dependency analysis method. The evaluation focuses on the precision of the dependency analysis method in identifying dependent entities.

5.5.1 Precision of the Dependency Analysis

Dependency analysis identifies all dependent entities that change together with a changing target entity. The main focus of our evaluation is to check whether the proposed dependency analysis method correctly identifies the necessary dependent entities that change based on the selected evolution strategy.

5.5.1.1 Experimental Evaluation

We extend the prototype (Section 4.6.1.1) and implement the dependency analysis algorithms. The prototype accepts a requested change operation, a target OCMS and a strategy to analyse the dependent entities of the target entity in the requested change. The prototype supports analysis of all types of dependencies for all types of entities. It analyses dependent entities for all applicable strategies that are selected by the user.

Before conducting the main evaluation, we conducted unit testing on the prototype to check the correct implementation of all the dependency rules and the robustness of the application. The prototype system passed through five iterations. In each of the iterations, we improve the implementation of the algorithm, corrected unidentified dependencies and fixed wrongly identified entities and dependencies.

For the purpose of the experiment, we classify entities into five strata representing their types. The five strata are classes, data properties, object properties, instances (including documents) and axioms. The classification enables us to observe the behaviour of the dependencies for each evolving entity in the OCMS. For each stratum, we selected 10 entities. The entities were selected randomly and proportionally from the three case studies (Appendix A, B and C) . We select entities from each case study because the OCMS in each case study has different behaviours. A total of 50 evolving entities are used in the evaluation process.

The evaluation was conducted as follows. First, we identified the dependent entities automatically using the prototype. Second, we identified all the dependent entities of all the selected entities manually. Third, we compared the results from the two methods. We used precision to measure the accuracy of the algorithm.

$$P(DA) = \frac{|CID|}{|CD|} \tag{5.1}$$

where:

$P(DA)$= Precision of the dependency analysis

$|CID|$= Number of correctly identified dependencies by the system

$|CD|$= Number of identified dependencies

We further measured the average time (in milliseconds) of the dependency analysis algorithm. Measuring the time allows us to estimate the response time of the application. The response time determines the usability of the system in real time scenario. We measured the time required to identify the dependent entities and their types. For each entity we conducted dependency analysis 10 times with a total of 500 observations. The experiment is conducted on a 3.00 GHz Intel(R) Core 2 Duo CPU with 4.00 GB of RAM, running on a 64-bit Windows 7 operating system.

5.5.1.2 Experimental Results and Discussion

A result of the dependency analysis conducted for one selected entity, *Activity*, is presented in Table 5.6. The table compares the analysis results of the automatic solution, expert solution and non-expert solution. The automatic solution identifies dependent entities following the proposed dependency analysis approach implemented in the prototype. The expert solution is conducted by an expert user and verified by other experts in ontology evolution with sufficient time and effort to identify the dependent entities. To avoid faults, the expert solution uses OWL API methods to list all axioms in the ontology. Then, the axioms related to the target entity are carefully examined to identify dependent entities. The non-expert solution examines the dependencies by ontology users who are not experts. The result shows that the automatic method identifies all the dependent entities which are identified by the expert solution. The automatic method further identifies all the dependencies identified by non-experts. Compared to the manual solution, the automatic method identifies more dependencies. This is mainly attributed to indirect dependent axioms that the manual analysis overlooked or failed to recognize. We also found that identifying the different dependency types using the manual method is time consuming.

Table 5.6: Comparison of the manual and automatic method

Entity	**Automatic Solution**				**Manual Solution**							
					Expert Solution				**Non-expert Solution**			
	TD	PD	DD	ID	TD	PD	DD	ID	TD	PD	DD	ID
Concepts	5	1	2	4	5	1	2	4	5	1	2	4
Axioms	14	0	5	9	14	0	5	9	9	0	5	4
Instances	1	1	2	0	1	1	2	0	1	1	2	0
Properties	0	0	0	0	0	0	0	0	0	0	0	0

TD = Total Dependent, PD = Partial Dependent
DD = Direct Dependent, ID = Indirect Dependent

We followed a similar analysis on individual entities to measure the overall precision of the proposed solution. The second Table 5.7 shows the average precision of the dependency analysis over the selected 50 entities. We followed the above approach to evaluate the accuracy of the proposed solution. We organized the precision based on dependency types.

Our dependency analysis algorithms enable us to achieve a 100% precision for the dependencies defined in this research. This result meets the minimum requirement we set for the dependency analysis phase. The dependency analysis includes useful information about the type of dependency, which is a crucial input for customization of change operations.

Table 5.7: Precision of OCMS dependency analysis (in 100%)

Entity	**TD**	**PD**	**DD**	**ID**	**Average**	**Time(ms)**
MeetingRoom	100%	100%	100%	100%	100%	15
Person	100%	100%	100%	100%	100%	19
Building	100%	100%	100%	100%	100%	15
hasCourseNumber	100%	100%	100%	100%	100%	17
hasName	100%	100%	100%	100%	100%	9
hasTitle	100%	100%	100%	100%	100%	7
hasMembers	100%	100%	100%	100%	100%	7
hasOffice	100%	100%	100%	100%	100%	6
hasWebsite	100%	100%	100%	100%	100%	6
takesCourse	100%	100%	100%	100%	100%	5
teaches	100%	100%	100%	100%	100%	5
CA106	100%	100%	100%	100%	100%	2
CA147	100%	100%	100%	100%	100%	2
Invent	100%	100%	100%	100%	100%	2
Janet	100%	100%	100%	100%	100%	2
Javed	100%	100%	100%	100%	100%	2
...						
Average	100%	100%	100%	100%	100%	10.3

The overall result of the dependency analysis process is encouraging and is suitable for the change impact analysis process.

Efficiency. The time on table 5.7 shows the average time (in milliseconds) the algorithm took to identify all dependent entities of an entity. To calculate the average time for analyzing the dependency of a single entity, we run the program 10 times. At each iteration, we record the time and calculated the average time at the end. For example, the time for analysing the dependent entities of the class ”meetingRoom” is 15ms. This means, if we run the application several times, the average time required will come closer to 15 minutes. In Table 5.7 it is clear that the time required to analyse the dependent entities of classes is more than the time for analysing dependent entities of instances. The overall average time for dependency analysis considers all entities including classes and instances. Thus from the experiment we calculated the overall average time for analysing dependent entities of any entity. This average time is 10.3 milliseconds. This means, for a given OCMS with similar size as the case studies, it takes an average of 10.3 ms to identify all the dependent entities.

The result is encouraging in that it provides fast response rate to the user and can be used in a real time environment. This result is dependent on the size of the OCMS in terms of the depth and breadth of the OCMS graph.

Time and Space Complexity. To provide a formal and general time and space requirement of the algorithm, which can be applied to any OCMS, we analysed and provided the efficiency using time and space complexity. In the worst case scenario, the algorithm exhibits $o(n^m)$ time complexity and $o(n^m)$ space complexity where n is the number of edges of a node and m is the depth of the graph.

The limitation in relation to complex classes is that the dependency analysis does not distinguish the dependency between entities in a complex class expression. This means, as complex classes are represented using axioms combining two or more classes, properties or restrictions, the analysis only considers the whole axiom as dependent entity. We do not focus on each of the constructs of the axiom and analyse the dependencies. At this stage, considering the whole axiom is sufficient to analyse the impacts. However, in the future, if we need to provide detailed explanation, we need to address this limitation.

5.6 Summary

The change impact analysis framework gives a high level description of the change impact analysis during evolution of the content, the annotation or the ontology. The change impact analysis process begins by capturing changes from the user and representing the changes using requested and generated change operations which are sufficient to fully implement the desired change. Generation of a complete change operation depends on the selected evolution strategy and the dependency which exists between the components of the OCMS.

To generate supplementary change operations, we need to understand the dependencies that exist among the entities in the OCMS. The target entity of the requested change and the change operation determine how the dependency analysis should be executed. The dependency analysis stage enables us to get all the entities that depend on the target entity. These dependent entities need to be changed accordingly due to the change in the antecedent entity. Thus, the dependency analysis process plays a major role in identifying entities that need to be changed in response to a requested change.

Determining what to do with the dependent entities is dictated by the evolution strategy. The evolution strategy tells us either to link the dependent entities to a parent entity, to delete them all or to leave it as it is. If we choose to link the dependent entities to another entity in the OCMS, the change operations we use include addition of new statements even if the user's change request is deletion of an entity.

The evolution strategy plays a key role in determining the number and the nature of the change operations generated in response to a given change request. The evolution strategies are further used to generate alternative change operations for comparing and selecting the different change impacts.

The evaluation result showed an encouraging result in the precision and efficiency of the dependency analysis method. The result meets the requirements of the change impact analysis phase.

Chapter 6

Change Impact Analysis Process

6.1 Introduction

In Chapter 5, we introduced the change impact analysis framework and its individual components. We further discussed the dependency analysis process and the evolution strategies. To understand the impacts of a requested change operation, we need to know the selected evolution strategy and identify the dependent entities accordingly. Thus, the output of the dependency analysis process serves as an input for the change impact analysis process. In this chapter, we discuss the change impact analysis process. The change impact analysis process covers both the analytical and constructive aspects. To cover the analytical aspects, we study the impacts of atomic, composite and domain-specific change operations. For the construction of the solution, we propose a bottom-up change impact analysis approach. This approach begins with the analysis of impacts of atomic change operations and moves up to composite and domain-specific change operations. Atomic change operations are organized to create composite change operations. Thus, the impacts of composite change operations are derived from the impacts of atomic change operations.

The change impact analysis process applies the following steps. First, we identify and characterize potential impacts of change operations. Second, we define the association between these impacts and atomic change operations. Third, we study and define the preconditions for the occurrence of each of the impacts. Finally, we assign the associated impacts to the atomic change operations when the preconditions are satisfied.

Although we can determine impacts of atomic change operations using this approach, it is not sufficient to analyse the impacts of composite and domain-specific change operations. The impacts of composite change operations may not be the same as a simple aggregation of impacts of the atomic change operations. There are impacts that cancel each other, impacts that balance or transform to another form. We investigate such behaviours and heuristically identify the associated rules. By using the change operations, the target entities and the parameters involved, we identify cancellation, balancing, and transformation rules. These rules are applied to identify impacts of composite and domain-specific changes.

This chapter is organized into seven different sections. Section 6.2 covers the change impact analysis process and its steps. Section 6.3 identifies structural and semantic impacts and the associated rules. Analysis of impacts of individual change operations and the algorithm used to assign impacts of individual change operations are discussed in Section 6.4. Analysis of impacts of composite change operations and the associated rules are given in Section 6.5. We evaluate the proposed solution for its accuracy and adequacy in Section 6.6. Finally, we give a summary of the chapter in Section 6.7.

6.2 Change Impact Analysis Process

The change impact analysis process begins by defining the possible impacts of changes in an OCMS system. Then, it uses change operations and their associated preconditions to identify the impacts. For each individual change operation, we perform a change impact analysis by considering the change operation, the target entity and the parameters [Abgaz & Pahl, 2012].

The change impact analysis process has two major steps. The first step is individual change impact analysis, which takes single atomic change operations and analyses their impacts individually. This process is done by matching the change operation with the structural and semantic impacts defined for atomic change operations. Once the match is found, the associated preconditions (Table 6.3) are checked. If the preconditions of the impacts

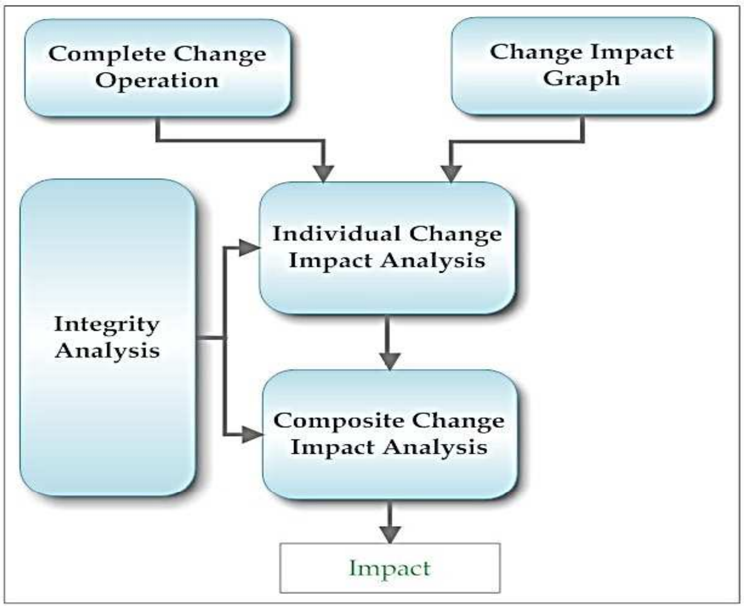

Complete Change Operation
Change Impact Graph
Integrity Analysis
Individual Change Impact Analysis
Composite Change Impact Analysis
Impact

Figure 6.1: Change impact analysis process

are satisfied, we assign the impacts and the affected entities to the change operation. We perform this process for all the atomic change operations contained in the complete change operations. A full discussion of atomic change impact analysis is presented in Section 6.4.

As part of the analysis, we identify change operations that have a potential to violate the satisfiability of the classes in the ontology, the consistency of the instances in the annotation or the null references to the content and the ontology layers. Our change impact analysis tool captures all individual change operations that introduce one or more impacts on the integrity of the OCMS and attach the impacts to the requested change operation for later processing.

The second step is composite change impact analysis, which takes the atomic change impacts and looks for impact cancellation, impact balancing and/or impact transformation of atomic change operations due to other change operations in the composite change. We use heuristics to analyse composite and domain-specific change operations. This phase analyses all the impacts of change operations when they are implemented together as a single change operation. Individual change operations that have a potential to introduce unsatisfiability and inconsistency are further checked, if other change operations are applied to resolve the inconsistencies created. Composite change impact analysis is discussed in Section 6.5.

Finally, we present the impacts of the change operations in two levels of detail. Depending on the users' requirement, we present either a summarized or a detailed analysis of the atomic and composite changes.

6.3 Impacts of Change Operations

Impacts of change operations in OCMS are diverse. We identify these impacts and investigate their characteristics. In this section, we discuss the impacts, their categories, the change operations that cause the impacts and the preconditions at which the impacts occur.

Impact: The term impact refers to the effect of change of entities due to the application of a change operation on one or more of the entities in the OCMS [Plessers et al., 2007] [Hassan et al., 2010] [Qin & Atluri, 2009]. Thus, a given atomic change operation (ACh) will have an impact $Imp : (ACh, P)$ if the associated precondition (P) is satisfied. The change impact analysis process uses a single change operation as an input at the atomic change operation level.

The impact function (Imp) is a function that maps an atomic change operation ACh to its corresponding impact whenever a given precondition P is satisfied.

Imp :$(ACh, P) \rightarrow (Impact)$

Where:

$Impact = StrImp \cup SemImp$

ACh = Atomic change, P = precondition

Imp :$(CCh) \rightarrow \{Imp : (ACh_1, P_1), Imp : (ACh_2, P_2), \ldots\ , Imp(ACh_n, P_n)\ \}$

Where:

$CCh = \{ACh_1, ACh_2, \ldots, ACh_n\}$

$ACh_1, \ldots ACh_n \in ACh$

CCh = Complete Change is a function of $ACh_1, \ldots, ACh_n$ + strategy

A complete change (CCh) is a composition of requested change and derived changes using a given evolution strategy.

6.3.1 Structural Impacts

Structural impacts are impacts that change the *structural dependency* (Section 5.3) between the entities. Structural impacts occur when we execute a change operation and if it impacts the structural dependency of entities in the OCMS. It can be caused by a deletion, addition or updating of an entity in the OCMS. The structural impacts of change operations and their associated rules are discussed below. The first four are adopted from [Stojanovic, 2004].

There are two types of structural impacts. The first type focuses on structural impacts that cause structural integrity violations. We call these impacts integrity-violating impacts. The second type focuses on changes, which are results or consequences of a given action. These are caused by changes that add or remove entities. We call these impacts integrity non-violating impacts. However, both are structural impacts of change operations.

Table 6.1: Structural impacts

No		Structural Impact	Acronym	Type
1	a	Addition + Orphan Class	(OC)	Integrity violating
	b	Deletion + Orphan Class	(OC)	Integrity violating
2	a	Addition + Orphan Instance	(OI)	Integrity violating
	b	Deletion + Orphan Instance	(OI)	Integrity violating
3	a	Addition + Property Cyclic Reference	(OPCR/DPCR)	Integrity violating
	b	Deletion + Property Cyclic Reference	(OPCR/DPCR)	Integrity violating
4	a	Addition + Class Cyclic Reference	(CCR)	Integrity violating
	b	Deletion + Class Cyclic Reference	(CCR)	Integrity violating
5	a	Addition + Null Reference to Content layer	(NRC)	Integrity violating
	b	Deletion + Null Reference to Content layer	(NRC)	Integrity violating
6	a	Addition + Null Reference to Ontology layer	(NRO)	Integrity violating
	b	Deletion+ Null Reference to Ontology layer	(NRO)	Integrity violating
7		Addition of new entity (AE)	(AC,AI,ADP,AOP,AA,AR)	Integrity non-violating
8		Deletion of existing entity (DE)	(DC,DI,DDP,DOP,DA,DR)	Integrity non-violating

A given change operation causes a structural impact in two ways. First, it either adds a new entity or removes an existing entity. Second, it violates the structural integrity of the OCMS. We use the following example to elaborate the situation.

In the first version (Figure 6.2.a), we can see that there are three entities. Due to a change operation $ADDClass(ArchivingEmail)$, the OCMS evolves to the second ver-

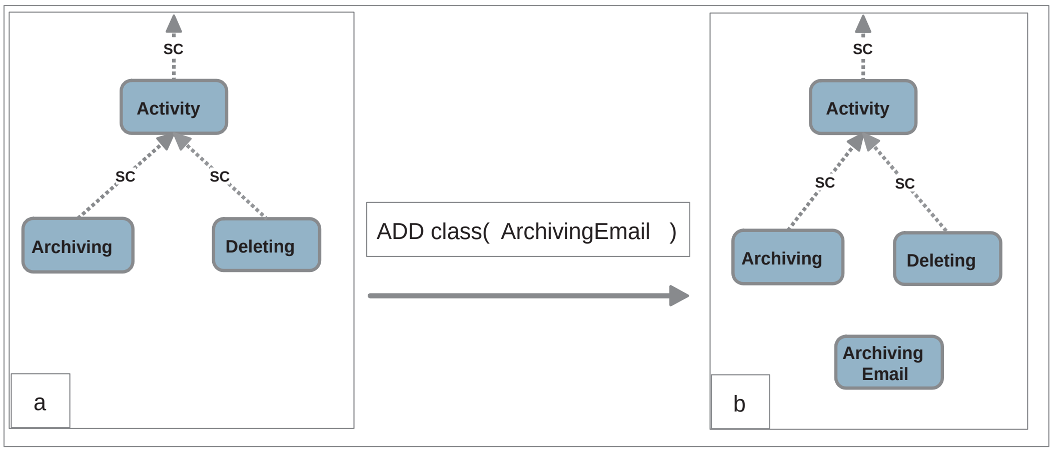


Figure 6.2: Example of structural impact

sion (Figure 6.2.b) which contains four entities.

When we compare the two versions, we can see the two impacts of the change operation. First, the change operation introduced a new class which was not available in the first version (Figure 6.2.a). Second, the change operation introduced an orphan class *(ArchivingEmail)*. Here, it is very important to distinguish between a change operation and the impact of a change operation. "Addition of new Entity (AE)" is an impact which is different from the $ADDClass(C)$ change operation, even if the impact is a straightforward consequence of the change operation. This distinction is important to clarify impacts independent of change operations. The separation is useful to systematically analyse impacts of composite change operations. The first impact is integrity non-violating, whereas the second impact is integrity violating impact.

To represent all the constructs of an ontology collectively, we use the term Entity (E). However, to refer to a specific constructs, we replace the term Entity (E) by Class (C), Data Property (DP), Object Property (OP), Instance (I), Axiom (A) and Restriction(R) whenever appropriate. The Structural impacts of ACh are:

$StrImp(ACh) = \{OC, CCR, OPCR, DPCR, OI, NRC, NRO, AE, DE\}$

where:

- **Orphan Class (OC)** occurs when a given class is introduced without a super class other than the default "Thing" class. Generally, OWL allows orphan classes, but sometimes the application requirements do not. It violates the concept-closure invariant, which states that every class node c_i in N, excluding the root class of the ontology, should have at least one super class c in N, giving closure to c_i : $\forall c_i \in N \setminus \{Root\} \wedge type(c_i) = \text{“}Class\text{”} \rightarrow \exists c \in N.\, CCDep(c_i, c)$.

- **Class Cyclic Reference (CCR)** occurs when a change operation introduces a cyclic reference to classes. It violates the class hierarchy invariant. The class hierarchy is a directed acyclic graph. For two class nodes c_1 and $c_2 \in N$, $\neg\exists c_1, c_2.\, CCDep(c_1, c_2)$ $\wedge\, CCDep(c_2, c_1)$.

- **Object Property Cyclic Reference ($OPCR$)** occurs when a change operation introduces a cyclic reference to object properties. It violates the property hierarchy invariant. The property hierarchy is a directed acyclic graph. For two object property nodes op_1 and $op_2 \in N$, $\neg\exists op_1, op_2.\, PPDep(op_1, op_2) \wedge PPDep(op_2, op_1)$.

- **Data Property Cyclic Reference ($DPCR$)** occurs when a change operation introduces a cyclic reference to data properties. It violates the property hierarchy invariant. The property hierarchy is a directed acyclic graph. For two object property nodes dp_1 and $dp_2 \in N$, $\neg\exists dp_1, dp_2.\, PPDep(dp_1, dp_2) \wedge PPDep(dp_2, dp_1)$.

- **Orphan Instance (OI)** occurs when a change operation introduces an instance with no link to a specific class. It violates the instance-closure invariant. Every instance node $i \in N$ is associated to a class node $c \in N$. such that $\forall i \in N, \exists c.\, CIDep(i, c)$.

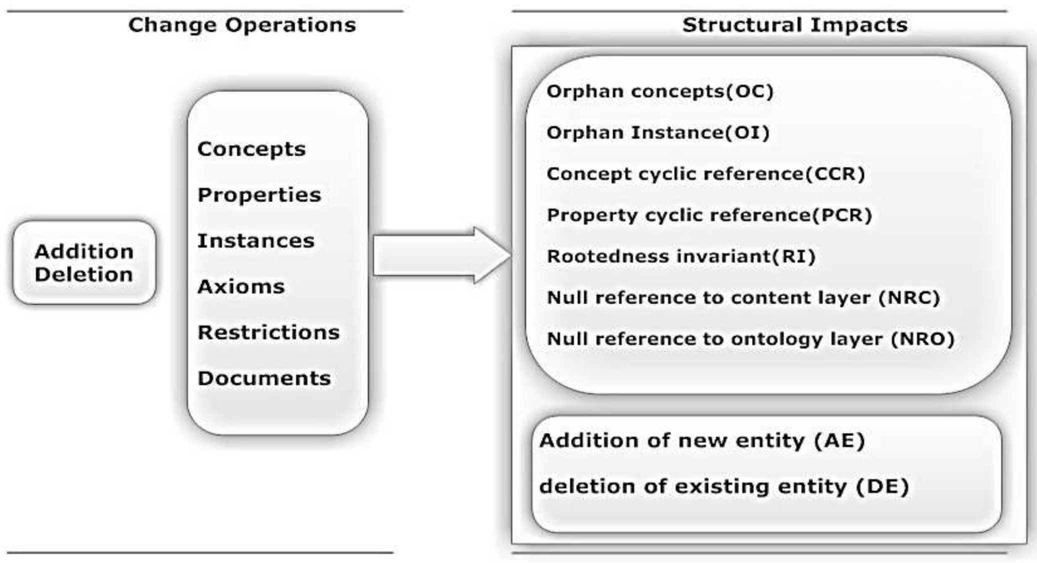
- **Null Reference to Content set (NRC)**. Every instance I in the annotation graph should have a corresponding document or part of document it refers in the content set. Given $G_A = (N_a, E_a)$, $\forall n_{a1} \in E_a.\, \exists n_{a1} \in Cont$ where $E_a = (n_{a1}, \alpha_a, n_{a2})$.

- **Null Reference to an Ontology layer (NRO)**. Every object node n_{a2} in the annotation graph should have a corresponding class in the ontology graph. Given $G_A = (N_a, E_a)$ and $G_o = (N_o, E_o)$, $\forall n_{a2} \in E_a.\, \exists n_{a2} \in N_o$ where $E_a = (n_{a1}, \alpha_a, n_{a2})$.

  Every instance property α_a in the annotation graph should have a corresponding property in the ontology graph. Given $G_A = (N_a, E_a)$ and $G_o = (N_o, E_o)$, $\forall \alpha_a \in E_a$.

$\exists \alpha_a \in N_o$ where $E_a = (n_{a1}, \alpha_a, n_{a2})$.

- **Addition of new Entity (AE)** occurs when any entity is added to the OCMS.
- **Deletion of new Entity (DE)** occurs when any entity is removed from the OCMS.



Change Operations
Addition
Deletion
Concepts
Properties
Instances
Axioms
Restrictions
Documents
Structural Impacts
Orphan concepts(OC)
Orphan Instance(OI)
Concept cyclic reference(CCR)
Property cyclic reference(PCR)
Rootedness invariant(RI)
Null reference to content layer (NRC)
Null reference to ontology layer (NRO)
Addition of new entity (AE)
deletion of existing entity (DE)

Figure 6.3: Structural impacts

The last two structural impacts directly correspond to the change operations and are straightforward. We consider them as impacts because they play a significant role during composite change impact analysis.

6.3.2 Semantic Impacts

Semantic impacts are impacts that change the semantics (interpretation) of entities in the OCMS. Whenever a structural change occurs, it causes a change on the meaning of the target entity or dependent entities. We identify existing semantic changes [Qin & Atluri, 2009] and derived semantic impacts from the changes. The semantic impact of an atomic change operation is defined as:

$SemImp(ACh) = \{EMD, ELD, PMR, PLR, AME, ALE, EG, ES, EInc, UE, IE\}$

where

$EMD =$ Entity More Described (class, property or instance)

$ELD =$ Entity Less Described (class, property or instance)

$PMR =$ Property More Restricted (object property, data property)

$PLR =$ Property Less Restricted (object property, data property)

$AME =$ Axiom More Expanded

$ALE =$ Axiom Less Expanded

$EG =$ Entity Generalized (class, property or instance)

$ES =$ Entity Specialized (class, property or instance)

$EInc =$ Entity Incomparable (class, property or instance)

$UE =$ unsatisfiable Entity (class, property)

$IE =$ invalid Entity (instance, instance property)

- **Entity More Described (**EMD**)** occurs when we add previously unknown facts about an entity. An entity node N_i is more described $EMD(N_i)$ by a change operation that transforms $G = (N, E)$ to $G' = (N, E')$ if $|edges(N_i) \in E'| > |edges(N_i) \in E|$. When the number of edges $E' \in G'$ containing N_i as a subject or as an object is greater than the number of edges $E \in G$ containing N_i as a subject or as an object, we say entity N_i is more described. This means, if there is a new edge added to a given entity, then that entity is more described. See Section 4.4.4 for details of getting the edge of a node.
- **Entity Less Described (**ELD**)** occurs when we remove an existing semantics (facts) about the entity. An entity node N_i is less described $ELD(N_i)$ by a change operation that transforms $G = (N, E)$ to $G' = (N, E')$ if $|edges(N_i) \in E'| < |edges(N_i) \in E|$. When the number of edges $E \in G$ containing N_i as a subject or as an object is greater than the number of edges $E' \in G'$ containing N_i as a subject or as an object, we say entity N_1 is less described. This means, if an existing edge is deleted from a

given entity, then that entity is less described.

- **Property More Restricted (**PMR**)** occurs when the existing semantics is more restricted. A property node $P \in N$ is more restricted $PMR(P)$ by a change operation that transforms $G = (N, E)$ to $G' = (N, E')$, for $E = (N_i, domainOf, P)$ and $E' = (N_j, domainOf, P)$, if $N_j \subset N_i$ or for $E = (N_i, rangeOf, P)$ and $E' = (N_j, rangeOf, P)$, if $N_j \subset N_i)$. If the domain class(N_j) of a given property is changed to a subclass of the original class (N_i), the property becomes more restricted. Likewise, if the range class(N_j) of a given property is changed to a subclass of the original class (N_i), the property becomes more restricted. A property more restricted shows a covariant property that converts the domain or the range of a property from a general class to a special class [Castagna, 1995].

- **Property Less Restricted (**PLR**)** occurs when the existing semantics is less restricted. A property node $P \in N$ is less restricted $PLR(P)$ by a change operation that transforms $G = (N, E)$ to $G' = (N, E')$, for $E = (N_i, domainOf, P)$ and $E' = (N_j, domainOf, P)$, if $N_i \subset N_j$ or for $E = (N_i, rangeOf, P)$ and $E' = (N_j, rangeOf, P)$, if $N_i \subset N_j)$. If the domain class(N_j) of a given property is changed to a super class of the original class (N_i), the property becomes less restricted. Likewise, if the range class(N_j) of a given property is changed to super class of the original class (N_i), the property becomes less restricted. A property less restricted shows a contravariant property that converts the domain or the range of a property from a special class to a general class [Castagna, 1995].

- **Axiom More Expanded (**AME**)** occurs when the axiom further extend its semantics to other entities. When a given axiom includes more entities and allows the semantics to apply for further entities, the axiom becomes semantically more expanded. An axiom E_i is more expanded $AME(E_i)$ by a change operation that transforms $G = (N, E)$ to $G' = (N, E')$, for $E = (N_i, \alpha, N_j)$ and $E' = (N'_i, \alpha, N_j)$ or $E' = (N_i, \alpha, N'_j)$, if $N'_i = N_i + N_k$ or $N'_j = N_j + N_k$ where $N_k \neq \emptyset$.

- **Axiom Less Expanded (ALE)** occurs when the axiom further restrict its semantics to fewer entities. When a given axiom excludes existing entities and restricts the semantics to apply for fewer entities, the axiom becomes semantically less expanded or more restricted. An axiom E_i is less expanded $ALE(E_i)$ by a change operation that transforms $G = (N, E)$ to $G' = (N, E')$, for $E = (N_i, \alpha, N_j)$ and $E' = (N'_i, \alpha, N_j)$ or $E' = (N_i, \alpha, N'_j)$, if $N'_i = N_i - N_k$ or $N'_j = N_j - N_k$ where $N_k \neq \emptyset$.

- **Entity Generalized (EG)** occurs when an entity become more general (move up in the hierarchy). Generalization occurs for structural relationships that define a parent-child relationship. An Entity node N_i is generalized $EG(N_i)$ by a change operation that transforms $G = (N, E)$ to $G' = (N, E')$, for $E = (N_i, \alpha, N_j)$ and $E' = (N_i, \alpha, N'_j)$, if $N_j \subset N'_j$ where $\alpha \in$ {*subClassOf, subDataPropertyOf, subObjectPropertyOf, instanceOf*} .

- **Entity Specialized (ES)** occurs when an entity become more specific (move down in the hierarchy). An Entity node N_i is specialized $ES(N_i)$ by a change operation that transforms $G = (N, E)$ to $G' = (N, E')$, for $E = (N_i, \alpha, N_j)$ and $E' = (N_i, \alpha, N'_j)$, if $N'_j \subset N_j$ where $\alpha \in$ {*subClassOf, subDataPropertyOf, subObjectPropertyOf, instanceOf*}.

- **Entity Incomparable ($EInc$)** occurs when a change on an entity is neither generalized nor specialized it. An Entity node N_i becomes incomparable $ES(N_i)$ by a change operation that transforms $G = (N, E)$ to $G' = (N, E')$, for $E = (N_i, \alpha, N_j)$ and $E' = (N_i, \alpha, N'_j)$, if $(N'_j \not\subset N_j) \wedge N_j \not\subset N'_j$ where $\alpha \in$ {*subClassOf, subDataPropertyOf, subObjectPropertyOf, instanceOf*}.

- **Unsatisfiable Entity (UE)** occurs when a change on a given entity introduces contradiction [Baader et al., 2003].

- **Invalid Entity (IE)** occurs when a change on a given instance or instance property introduces invalid interpretation [Qin & Atluri, 2009].

Researchers [Stojanovic, 2004] [Qin & Atluri, 2009] have already defined some semantic changes in ontologies. In this research, we extend the semantic changes to identify semantic impacts of change operations. However, we customized existing ones and introduced new impacts for applicable entities. The expanded impacts derived from the above semantic changes are discussed in Table 6.2.

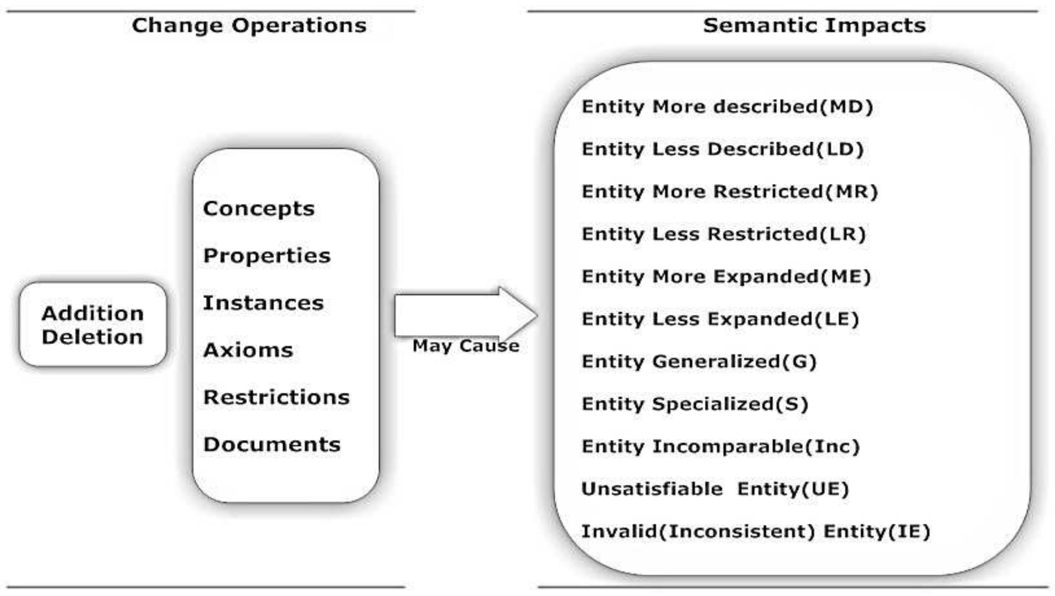


Figure 6.4: Semantic impacts

Semantic impacts are caused by structural changes [Qin & Atluri, 2009]. Some of the structural changes, which involve axioms that specify relationships between classes (subclass of, intersectionOf, disjointWith, complementOf) and relations between properties and classes (domain, range) may cause semantic impacts.

The impact analysis process identifies one or more of the above structural or semantic impacts of the requested change operation. The change operation may make the dependent entity an orphan entity. Two or more change operations can also cause generalization or specialization of the dependent entities.

Table 6.2: Semantic impacts

No.	Semantic Impact	Acronym
1	Entity More Described (EMD)	(CMD,DPMD,OPMD,IMD)
2	Entity Less Described (ELD)	(CLD,DPLD,OPLD,ILD)
3	Entity More Restricted	(OPMR)
4	Entity Less Restricted	(OPLR)
5	Entity More Expanded	(AME)
6	Entity Less Expanded	(ALE)
7	Entity Generalized (EG)	(CG,DPG,OPG,IG)
8	Entity Specialized (ES)	(CS,DPS,OPS,IS)
9	Entity Incomparable (EI)	(CInc, DPInc, OPInc, IInc)
10	Unsatisfiable Class/Property (UE)	(UC,UDP,UOP)
11	Invalid Instance/Instance Property (IE)	(II, IIP)

6.3.3 $\mathcal{AB}$**ox versus** $\mathcal{TB}$**ox Impacts**

Impacts of a change operation can be viewed from the perspective of the kind of statement they affect. In description logic, we can classify statements into $\mathcal{TB}$ox and $\mathcal{AB}$ox [Horrocks, 2003]. The statements that focus on the ontology terminology are the $\mathcal{TB}$ox statements and the statements that focus on the instances (individuals) in the annotation layer are the $\mathcal{AB}$ox statements. Change operations may have an impact on the $\mathcal{AB}$ox statements or on the $\mathcal{TB}$ox statements. A separate treatment of the statements is important to analyse impacts of the operations. It further helps us filter the statements that are affected by the change operation.

$\mathcal{TB}$**ox Impacts.** $\mathcal{TB}$ox statements are affected by operations that change the axioms related to the terminology in the ontology. The terminology box is defined as statements that assert how classes or roles are related to each other [Baader et al., 2003]. The impact of such change operations revolve around the satisfiability and the coherence of the ontology. This means, a change operation may violate the semantic integrity of the terminology.

$\mathcal{AB}$**ox Impacts.** $\mathcal{AB}$ox statements are affected by operations that change the axioms related to annotation instances or individuals in the assertion box. Assertion box statements, in our case, are defined as statements related to instances and instance assertions. A change on the assertion statements may result inconsistency due to contradictory statements about the instances [Horrocks, 2003] or invalidity of the interpretation of an instance with respect to an ontology [Qin & Atluri, 2009].

6.3.4 Addition versus Deletion Impacts

Impacts of addition and deletion operations are discussed under structural impacts. In this section we discuss addition and deletion as change operations from a perspective of the effort and resource required to automatically implement them. In terms of change operations, adding a new entity and deleting an existing entity involves different procedures and resources.

Impacts of Addition. The addition operation introduces a new ontology entity, annotation entity or content entity in the OCMS. If a new entity is introduced in the OCMS, it can be a class, a property, an instance, an axiom, etc. Such kinds of changes introduce new knowledge and may require an update of the overall system. The addition of a new entity involves creating that new entity. If the new entity needs to be linked with existing entities, we need to search and find those entities in the OCMS. This involves extra resource for searching. However, addition operation usually attaches the new entity on the OCMS.

Impacts of Deletion. A deletion operation removes existing entities from the OCMS. The deletion operation in the ontology layer removes an existing class, property, axiom or restriction. The deletion operation in most of the cases is the source of cascaded impacts. The impact of the operation becomes complex depending on the position of the deleted entity. When we delete an entity, we should search its usage in the OCMS and delete all instances of usage. This involves searching the whole graph and deleting every instance of the entity.

Compared to addition operation, deletion operation requires more time and resource. If the deletion operation uses the cascade strategy, it requires more resources and time to complete the execution of the change operation.

6.4 Individual Change Impact Analysis

Individual change impact analysis takes individual change operations and analyses their impacts. The individual change impact analyses the atomic changes and assigns impacts if they satisfy the preconditions. We identify atomic change operations categorized by the type of operations and target entities. We analyse their potential impacts and identify the preconditions of each impact in Section 6.4.1. To analyse the impact of an atomic change, we search the change operation and read the associated impact and its precondition. We check if the precondition for the impact is satisfied in the OCMS. Whenever the precondition is satisfied, the impact is assigned to the change operation.

If associated preconditions are defined for the change, we check the preconditions. If the preconditions are satisfied, we take the impact and the target entity as an impact of the change operation. For semantic impacts that cause unsatisfiability or inconsistency, the individual change operation may not be the sole reason for the unsatisfiability or inconsistency. In such situations, we keep traces of those statements for explaining the reason for the violation of the integrity and for resolving the problem. If the preconditions are not satisfied, we move to the next impact defined for the change operation and continue the above process until we finish all the atomic change operations contained in the complete change.

6.4.1 Impacts of Atomic Change Operations

We identified different atomic change operations and studied their semantic and structural impacts in Section 6.3. To discuss atomic change impact analysis, we take frequently observed [Goncalves et al., 2011] change operations. The list of the impacts of the other atomic change operations and their preconditions is given in Table 6.3.

- The structural impact of *Add Class*(c_1) is the *addition of new class* $AC(c_1)$ and that *class being orphan* $OC(c_1)$. $StrImp(AddClass(c_1)) = AC(c_1)$ and $OC(c_1)$. When a new class is added by the *Add Class* operation, it means we introduced a new class and that class becomes an orphan class. This is because this single operation does a single task of adding a new class.

- The structural impact of *Add SubClass(c_1, c)* is the *addition of a new axiom* AA $(FullAxiom)$[1] and if $\exists c.\ CCDep(c, c_1)$, this change introduces a cyclic reference. This is defined as a precondition for this atomic change operation and if it is satisfied, then we can assign a cyclic reference to both classes $CCR(c_1)$ and $CCR(c)$. If the precondition is false, the change does not introduce a cyclic reference impact.

  The semantic impact of this change operation makes the two classes more described (CMD). This means now we know that c_1 is a special class of C and we know more information about the two classes. Thus, the semantic impacts are $CMD(c_1)$ and $CMD(c)$.

- The structural impact of *Delete Class(c)* is the *deletion of an existing class* $DC(c)$, $StrImp(DeleteClass(c)) = DC(c)$. The semantic impact of the change operation is $SemImp(DeleteClass(c)) = UA(e_i)$ if $\exists e_i.\ CADep(e_i, c)$. This means when a class is deleted, all axioms that refer to this class will be impacted and may need to be deleted or modified.

- The structural impact of *Delete SubClass(c_1, c)* is the *deletion of an existing axiom* $DA(FullAxiom)$ and an *Orphan Class*(c_1), if the class c is the only super class of c_1. Thus, $StrImp(DeleteSubClass(c_1, c)) = DA(fullAxiom)$ and $OC(c_1)$ if $\exists c.$ $CCDep(c_1, c) \wedge \neg \exists d.\ CCDep(c, d) \wedge c \neq d$.

  The semantic impact of the change operation is that both classes become less described due to the removal of an existing fact about the two classes. Thus, $SemImp$ $(Delete\ SubClass(c_1, c)) = CLD(c_1)$ and $CLD(c)$.

- The semantic impact of *Add DisjointClasses* (c_1, c_2) is:

  – an *unsatisfiable class* (UC) if there exists a class, which is in a subClass hierarchy of both c_1 and c_2. $SemImp(AddDisjointClass(c_1, c_2)) = UC$ if $\exists c : C, c \in c_1^* \wedge c \in c_2^*$

[1]Full Axiom represents the entire axiom, not individual entities constructing the axiom

– an *invalid instance* (II), if there exists an instance which is an instance of the class c_1 and c_2 or the subclass of c_1 and c_2. $SemImp(AddDisjointClass(c_1, c_2))$ = II if $\exists i : (i, c_1^*) \wedge (i, c_2^*)$. c^* represents all the super classes in the hierarchy.

Table 6.3: Potential impacts of selected atomic change operations

No	Change Operation	Impact Type	Impact (Entity)	Impact Precondition
1	Add Class (c)	Structural	AC(c), OC(c)	None
2	Add SubClass (c_1, c)	Structural	AA (FullAxiom)	None
			CCR(c_1), CCR (c)	$\exists c$. **CCDep**(c, c_1)
		Semantic	UC (c_1)	$\exists c_1$. **CCDep**$(c_1, d) \wedge$ disjointClasses(c, d)
			CMD(c_1), CMD(c)	None
3	Delete Class (c)	Structural	DC (c)	None
		Semantic	UA (a_i)	$\exists a_i$. **CADep**(a_i, c)
4	Delete SubClass (c_1, c)	Structural	DA (FullAxiom)	None
			OC (c_1)	$\exists c_1$. **CCDep**(c_1, c) $\wedge\neg\exists c_1$. **CCDep**$(c_1, d) \wedge c \neq d$
		Semantic	$CLD(c_1)$, CLD (c)	None
5	Add Instance (i)	Structural	AI(i), OI(i)	None
6	Add InstanceOf (i, c)	Structural	AA (FullAxiom)	None
		Semantic	II (i)	$\exists c_1$. **CIDep**$(i, d) \wedge$ disjointClasses(c, d)
			IMD(i), CMD(c)	None
7	Delete Instance (i)	Structural	DI (i)	None
		Semantic	UA (a_i)	$\exists a_i$. **IADep**(a_i, i)
8	Delete InstanceOf (i, c)	Structural	DA (FullAxiom)	None
			OI (i)	$\exists i$. **CIDep**(i, c) $\wedge\neg\exists d$. **CIDep**$(i, d) \wedge c \neq d$
		Semantic	$ILD(i)$, CLD (c)	None
9	Add ObjectProperty (op)	Structural	AOP(op)	None
10	Delete ObjectProperty (op)	Structural	DOP (op)	None
		Semantic	UA (a_i)	$\exists a_i$. **PADep**(a_i, op)

			IIP(ip)	$\exists ip$. **PIPDep**(ip, op)
11	Add SubObjectProperty (op_1, op)	Structural	AA (FullAxiom)	None
			OPCR(op_1), OPCR(op)	$\exists op$. **PPDep**(op, op_1)
		Semantic	UOP (op_1)	$\exists op_1$. **PPDep**$(op_1, oq) \wedge$ disjointObjectProperty(op, oq)
			OPMD(op_1), OPMD(op)	None
12	Delete SubObjectProperty (op_1, op)	Structural	DA (FullAxiom)	None
		Semantic	$OPLD(op_1)$, OPLD (op)	None
13	Add DataProperty (dp)	Structural	ADP(dp)	None
14	Add SubDataProperty (dp_1, dp)	Structural	AA (FullAxiom)	None
			DPCR(dp_1), DPCR (dp)	$\exists dp$. **PPDep**(dp, dp_1)
		Semantic	UDP (dp_1)	$\exists dp_1$. **PPDep**$(dp_1, dq) \wedge$ disjointDataProperty(dp, dq)
			OPMD(dp_1), OPMD(dp)	None
15	Delete DataProperty (dp)	Structural	DDP (dp)	None
		Semantic	UA (a_i)	$\exists a_i$. **PADep**(a_i, dp)
			IIP(ip)	$\exists ip$. **PIPDep**(ip, dp)
16	Delete SubDataProperty (dp_1, dp)	Structural	DA (FullAxiom)	None
		Semantic	$OPLD(dp_1)$, OPLD (dp)	None
17	Add Disjoint $Class(c_1, c_2)$	Structural	AA (FullAxiom)	None
		Semantic	UC (c_1), UC(c_2)	$\exists c$. **CCDep**$(c, c_1)\wedge$ **CCDep**(c, c_2)
			II(I)	$\exists i$. **CIDep**$(i, c_1)\wedge$ **CIDep**(i, c_2)
			$CMD(c_1), CMD(c_2)$	None
18	Add Equivalent $Class(c_1, c_2)$	Structural	AA (FullAxiom)	None
		Semantic	UC (c_1), UC(c_2)	$\exists c$. **CCDep**$(c_1, c)\wedge$ **CCDep**$(c_2, d)\wedge$ DisjointClasses(c, d)
			$CMD(c_1), CMD(c_2)$	None
			II(I)	$\exists i$. **CIDep**$(i, c_1)\wedge$ **CIDep**(i, c_2)

Analysing the semantic and the structural impacts of atomic change operations requires a careful analysis of all possible scenarios. We use different cases to identify the scenarios. This approach is time consuming, but it is a one time task. The other main advantage of this approach is that, it is very fine-grained and it can be used to process the impacts of any composite and domain-specific changes composed of atomic change operations. Once we define the potential impacts of atomic change operations and the conditions at which the impacts occur, the next step is to use them as an input to determine the actual impacts of change operations when an OCMS evolves.

6.4.2 Steps for Individual Change Impact Analysis

The individual change impact analysis process is done step by step. The steps are outlined here and discussed with a flowchart (Figure. 6.5) and an example in detail.

1. Select an individual change operation from a complete change
2. Search the corresponding change operation in the potential impacts of change operations
3. Read its structural impacts
4. Assign the structural impact to the affected entity if the preconditions are satisfied
5. Read its semantic impacts
6. Assign the semantic impact to the affected entities if the preconditions are satisfied

We will explain each of the steps in detail. We will use the case study described in appendix A. Input: Requested change (consider the following complete change containing two atomic changes)

```
1. Delete  SubClassOf (DeletingFile,   Deleting)
2. Delete  Class  (DeletingFile)
```

Step 1. Get a change operation from a complete change. Take the operation (Delete) and the target entity (subClassOf).

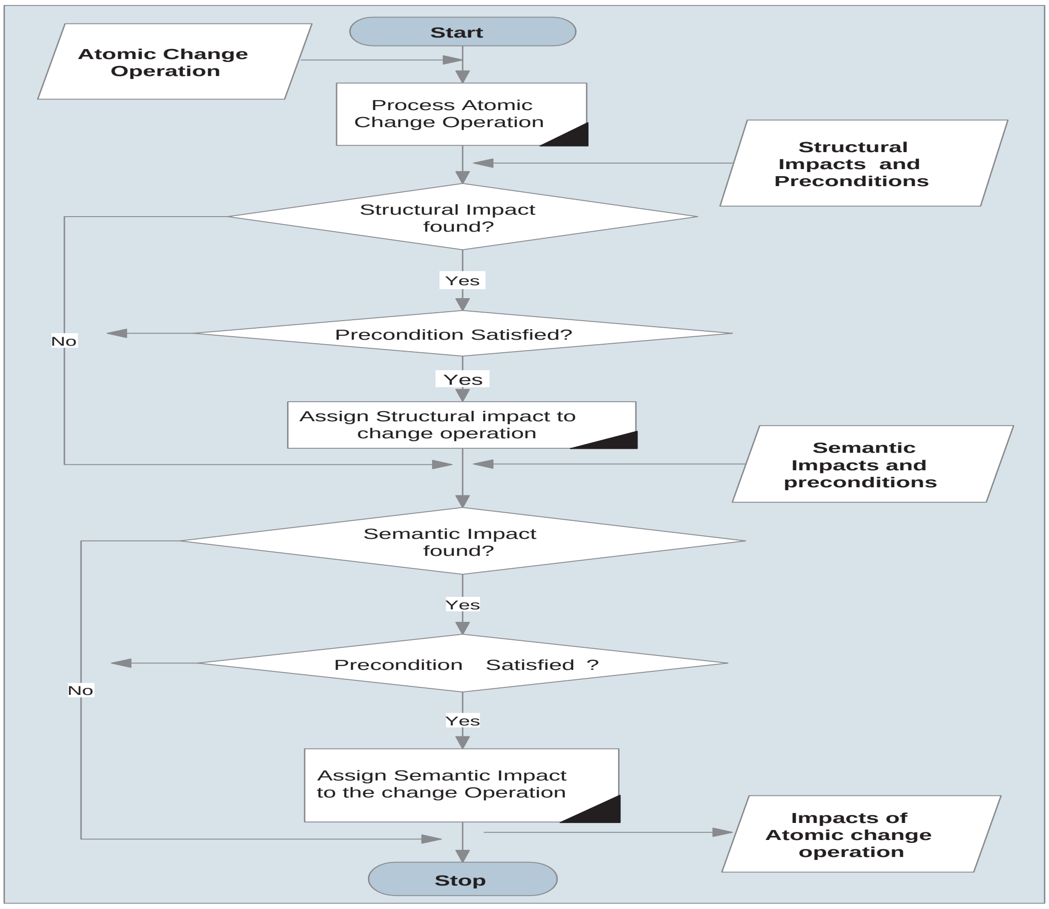

Start
Atomic Change Operation
Process Atomic Change Operation
Structural Impacts and Preconditions
Structural Impact found?
Yes
No
Precondition Satisfied?
Yes
Assign Structural impact to change operation
Semantic Impacts and preconditions
Semantic Impact found?
Yes
No
Precondition Satisfied ?
Yes
Assign Semantic Impact to the change Operation
Impacts of Atomic change operation
Stop

Figure 6.5: A flowchart: atomic change impact analysis

Step 2. Search the corresponding change operation in the potential impacts of change operations. From the potential impacts of change operations, find the corresponding change operation.

Step 3. Read the structural impact. The structural impact of the change operation is the Deletion of an Axiom (DA). The affected entity is the full axiom. The second structural impact of the change operation is the introduction of an Orphan Class (OC).

Step 4. Check the structural impact precondition. This impact, however, has a precondition that we need to check. If *DeletingFile* is totally dependent on *Deleting*, then the implementation of this operation introduces orphan class (OC) for the first entity. The structural impact parameter shows the entity which is structurally affected. If the precondition is not satisfied, we simply ignore this impact.

Step 5. Read the semantic impact. The change operation makes the two classes to be less described. This is because of a removal of the axiom, which carries semantic information about both classes. With the presence of the axiom, we know that *DeletingFile* is a subclass of *Deleting*, i.e., now we know less about the two entities. This impact applies for both entities (DeletingFile and Deleting).

Step 6. Check the semantic impact precondition. There is no semantic impact precondition associated with this change operation. This means this impact occurs whenever the change operation is implemented. When a subclass of an axiom that links two classes is removed, the two classes become less described. The output of this phase is the original change request together with the structural and semantic impacts attached.

Table 6.4: Impact analysis output

1	ChangeOperation	Delete
2	TargetEntity	subClassOf
3	Parameter Type	Class
	Position	0
	Value	http://CNGL.ie#DeletingFile
4	Parameter Type	Class
	Position	1
	Value	http://CNGL.ie#Deleting
5	ReasoningType	$\mathcal{TB}$ox
6	StructuralImpact	DA
7	StructurallyAffectedEntity	FullAxiom
8	StructuralImpact	OC
9	StructurallyAffectedEntity	http://CNGL.ie#DeletingFile
10	SemanticImpact	CLD
11	SemanticallyAffectedEntity	http://CNGL.ie#DeletingFile
12	SemanticImpact	CLD
13	SemanticallyAffectedEntity	http://CNGL.ie#Deleting

When we check the preconditions, we also keep track of the entities participating in the change operation. This information is used for further analysis of the integrity of the OCMS. For example, if we are making two classes equivalent and if there exists a disjoint axiom involving the classes, the semantic impact becomes unsatisfiable class (UC). Thus, we keep track of such statements and store them for the composite change impact analysis phase.

6.4.3 Algorithm for Individual Change Impact Analysis

A general algorithm that attaches the structural and the semantic impacts of change operations is given below. The algorithm takes the complete change operation, analyses the impacts of the atomic change operations and returns the associated impacts of the change operations.

Algorithm 6 Assign Individual Change Impacts $(CCh, Impact)$

```
1:  Input : Complete Change operation $(CCh)$, Change impacts(Impact)
2:  Output: Complete Change operation with impacts
3:  for each atomic change operation$(ACh)$ in $CCh$ do
4:      if $ACh$ is found in change impacts then
5:          assign corresponding impact to Imp
6:          for each strImp in Imp do
7:              if structural precondition(imp)=true then
8:                  attach the affected entity to the strImp
9:                  attach strImp to $ACh$
10:             end if
11:         end for
12:         for each semImp in Imp do
13:             if semantic precondition(imp)=true then
14:                 attach the affected entity to the semImp
15:                 attach semImp to $ACh$
16:             end if
17:         end for
18:     end if
19: end for
20: return $CCh$
```

The change impact analysis approach at this stage is compliant with existing tools. A tool that generates change operations at an atomic level can exploit the individual change impact analysis step and can find both the structural and the semantic impacts of the individual changes. Individual change impact analysis generates the impacts of atomic change operations individually and gives us crucial information about the impacts. However, when changes are applied in a batch as a composite change operation, the impact of one change operation depends on the other change operations. Individual change impact analysis yield detailed impacts of atomic change operations. But it does not consider the previous or the following change operations. The impact of a composite change operation is not a simple collection of the impacts of the atomic change operations contained in it. Thus, we require a different impact analysis strategy at a composite level.

6.5 Composite Change Impact Analysis

Composite change impact analysis focuses on analysing impacts of two or more change operations when they are executed together. The independent analysis of an atomic change operation within a complete change representation indicates only the impacts of that specific atomic change operation. When we implement a requested change, we may have more than one change operation to fully implement the requested change. In such a situation, to understand and to find out the impacts of the complete change operation, as a single transaction, we need to go further to composite change impact analysis. Composite change impact analysis considers the impacts of one change operation in relation to impacts of other preceding or following change operations. When a composite change operation is implemented, the impacts of the composite change may not be the same as the aggregation of the impacts of its constituent individual atomic change operations. The impacts may reduce or be transformed to other impacts. Composite change impact analysis identifies techniques to analyse the impacts of composite change operations. To analyse these impacts, we employ novel techniques, such as impact cancellation (Section 6.5.1), impact balancing (Section 6.5.2) and impact transformation (Section 6.5.3), that exploit dependencies between individual changes and impacts.

6.5.1 Impact Cancellation

Impact cancellation applies for two change operations. Impact cancellation occurs when the impact of one operation cancels or overrides the impact of the other operation on a given entity. This means, the impact of a given change operation removes the impacts caused by another change operation, or one impact subsumes the other impact. Impact cancellation mainly focuses on impacts caused by similar operations. It occurs between two additions or between two deletions. For example, if the impact of one change operation introduces an *Orphan Entity (OE)* and a following change operation deletes the orphan entity resulting in a structural impact *Delete Entity (DE)*, then the impact of the second change operation overrides the impact of first change operation. This means, the orphan entity is deleted by the second change operation. In this case, we remove the impact of the first change operation (*Orphan Entity (OE)*) because that entity is deleted.

Identification of change impacts that cancel each other requires analysing the impacts of the change operations when they are applied on a single entity. We analyse the impacts, the operations, the target entities, affected entities, and the order at which they appear in the parameter and in the complete change operation. By doing so, we identify associated change operations and impacts that are candidates to impact cancellation. Impact cancellation uses the following rules to identify and cancel impacts of composite change operations. The rules are derived from our observation of the case studies and validated using experiments.

- **Rule 1.** When a target entity is affected by an operation ACh_1, and if that target entity is deleted by another operation OP_2, the applicable structural and semantic impacts of OP_1 on the target entity will be cancelled.

  For $CCh = \{ACh_1, ACh_2\}, Imp : \{CCh\}$ = $Imp\{ACh_2\}$ if
  $Imp\{ACh_1\}$ = $strImp(x) \cup (semImp(x) \backslash DE(x)) \wedge$
  $imp\{ACh_2\} = DE(x)$.

- **Rule 2.** When a change operation ACh_1 is executed, if it introduces an impact (I_1), but if there is another change operation ACh_2 that changes the precondition of the impact (I_1), the impact (I_1) will be cancelled.

  For $CCh = \{ACh_1, ACh_2\}, Imp\{ACh_1, ACh_2\}$ = $Imp\{ACh_2\}$ if
  $Imp\{ACh_1\}$ = $OE(x) \wedge imp\{ACh_2\} = AA(\alpha)$
  where $\alpha = (x, subclassOf, y) \vee (x, instanceOf, y)$.

We further identify pairs of cancelling and cancelled impacts for the two rules. The following table gives the pairs of impacts that are candidates for cancellation. In the first rule, if an entity is deleted, all the structural and semantic impacts associated with it will be removed. In the second rule, we remove orphan entities when the following change operations add an axiom that links the entity to a parent entity.

Table 6.5: Candidate impacts for cancellation

Rules	Cancelling Impact	Candidates for cancellation
Rule 1	*Delete Entity(DE)*	All StrImp except *DE*
		All SemImp
Rule 2	*Axiom Added(AA)*	*OE*

A typical characteristics of cancellation is that the change operations, that have cancelling impacts, have the same operation (addition and addition or deletion and deletion), but one acts on a node (e.g. class) and another on the edge (e.g. subclass) linked to that node. The rationale behind impact cancellation is to filter out impacts, which are subsumed by other impacts. In composite change impact analysis, keeping the impacts of an entity, which is totally removed or overridden by another impact, is meaningless.

The following example elaborates how the impact cancellation process is used to analyse impacts of composite change operations. The impact of *Delete SubClass (DeletingFile, Deleting)* and *Delete Class (Deleting)* is given in Table 6.6. The two atomic change operations are candidates for impact cancellation according to Rule 1. The target class *DeletingFile* is affected by the first change and is deleted by the second change operation.

Table 6.6: Impact cancellation using Rule-1

No	Change Operation	Structural Impact	Semantic Impact
1	*Delete SubClassOf* (*DeletingFile*,*Deleting*)	*OC(DeletingFile*)	*CLD(DeletingFile*) *CLD(Deleting*)
2	*Delete Class (DeletingFile)*	*DC(DeletingFile*)	None
	After Cancellation		
1	*Delete SubClassOf (DeletingFile, Deleting)*	None	*CLD(Deleting*)
2	Delete Class *(DeletingFile*)	*DC(DeletingFile)*	None

If we look at the two change operations, the first change operation deletes the SubClassOf axiom and introduces the *OC(DeletingFile*) impact. However, the following change operation deletes the class *DeletingFile*. The first change operation makes the *DeletingFile* class an orphan class and semantically less described. The second change operation removes the class from the ontology layer. Thus, the *OC(DeletingFile)* and *CLD(DeletingFile)* impacts are cancelled from the first operation.

To elaborate the impact cancellation process further, let us take two atomic change operations *Add Class(GUI)* and *Add subClassOf(GUI, UserInterface)*. The first change operation introduces an orphan class *OC(GUI)*. However, the second operation falsifies the precondition of orphan class impact by introducing an axiom that link the orphan class to *UserInterface* class. Thus, the newly added axiom *AA(FullAxiom)*, which is *subClassOf(GUI, UserInterface)*, overrides the orphan class impact and removes it from the list. The impacts are reduced from 5 to 4 because the *OC(GUI)* impact is removed. The impacts before and after cancellation are depicted in Table 6.7

Table 6.7: Impact cancellation using Rule-2

No	Change Operation	Structural Impact	Semantic Impact
1	*Add Class(GUI)*	*OC(GUI)*	None
		AC(GUI)	
2	*Add SubClass(GUI, UserInterface)*	AA(FullAxiom)	*CMD(GUI)*
			CMD(UserInterface)
After Cancellation			
1	*Add Class(GUI)*	*AC(GUI)*	None
2	*Add SubClass(GUI, UserInterface)*	*AA(FullAxiom)*	*CMD(GUI)*
			CMD(UserInterface)

6.5.2 Impact Balancing

The impacts of two change operations balance each other when one change operation introduces an impact to an entity and another change operation removes the impact from the entity. Unlike impact cancellation, impact balancing only occurs between an addition and a deletion operation with the same target entity (e.g. class with class and subclass with subclass). The main difference between balancing and cancelling is that balancing always occurs either between two structural impacts or between two semantic impacts. However, in the case of cancelling, a structural impact cancels both structural impacts and semantic impacts. To facilitate impact balancing, we identify counter-impacts for the candidate impacts.

- **Rule 3.** When a given change operation (ACh_1) affects the target entity with an impact, and when another change operation (ACh_2) affects the same entity with a counter-impact or vice versa, the two impacts may balance each other.

  $Imp\{ACh_1, ACh_2\}$ = $\emptyset$ if

  ($Imp\{ACh_1\}$ = $EMD(x)$ $\wedge$ $Imp\{ACh_2\}$= $ELD(x)$) $\vee$

  ($Imp\{ACh_1\}$ = $AME(x)$ $\wedge$ $Imp\{ACh_2\}$= $ALE(x)$) $\vee$

  ($Imp\{ACh_1\}$ = $OPLR(x)$ $\wedge$ $Imp\{ACh_2\}$= $OPMR(x)$) $\vee$

  ($Imp\{ACh_1\}$ = $AE(x)$ $\wedge$ $Imp\{ACh_2\}$= $DE(x)$).

Impact balancing is commutative. This means, $Imp\{ACh_1, ACh_2\}$ =$Imp\{ACh_2, ACh_1\}$.

Table 6.8: Candidate impacts for balancing

Impacts	**Counter-Impacts**
Entity More Described (EMD)	Entity Less Described (ELD)
Axioms More Expanded (AME)	Axioms Less Expanded (ALE)
Object Property Less Restricted (OPLR)	Object Property More Restricted (OPMR)
Addition of new Entity (AE)	Deletion of existing Entity (DE)

For example, if a change operation deletes an existing entity, but that same entity is added later or vice versa, the impacts of the two change operations can balance each other. An axiom (b $subClassOf$ c) added to make a class (b) a subclass of another class (c), and an axiom (c $subClassOf$ b) deleted to break the subclass relationship between the class (c) and its previous superclass (b) can balance each other. In the example below, the first operation introduced “cyclic class hierarchy”, but the second introduced “orphan class”, since these two change operations act on the same class, the impact can be balanced and both cyclic and orphan impacts can be removed.

To explain the impact balancing process, let us take two atomic change operations: *Add SubClassOf(DeletingFile, Activity)* and *Delete SubclassOf(DeletingFile, Deleting)* are candidates for balancing. The *Add SubClass* matches *Delete SubClass* and the class *DeletingFile* is a common entity in both operations. When we view these two change operations together, they show a change in the subclass hierarchy of *DeletingFile* from *Deleting* to *Activity*. Thus, we can say that the subclass of an axiom is modified and we understand that the addition followed by deletion is just a modification. DeletingFile is more described first and less described next, thus the semantic impacts *CMD* and *CLD* balance each other, and thus both of them will be removed. The *Add Axiom* and the *Delete Axiom* impacts are also balanced, thus will be removed. However, we can see that the class *Activity* is more described (CMD) and the class *Deleting* is less described (CLD). This impact reflects what is happening to the two classes and we do not balance the two impacts because they affect different entities.

Table 6.9: Impact balancing using Rule-3

No	Change Operation	Structural Impact	Semantic Impact
1	*Add SubClassOf (DeletingFile, Activity)*	*AA(FullAxiom)*	*CMD(DeletingFile)* *CMD(Activity)*
2	*Delete SubClassOf (DeletingFile, Deleting)*	*DA(FullAxiom)*	*CLD(DeletingFile)* *CLD(Deleting)*
After Balancing			
1	*Add SubClassOf (DeletingFile, Activity)*	None	*CMD(Activity)*
2	*Delete S subClassOf (DeletingFile, Deleting)*	None	*CLD(Deleting)*

After balancing of the above two change operations, we remove the CLD and CMD semantic impacts and the AA and AD structural impacts. However, when two change impacts balance each other, they introduce a higher level change impact, which is caused by composite change operations. The change operations may introduce impacts such as specialization or generalization of the entities, more restriction or less restriction on cardinalities of properties, etc. Thus, the original change impacts are transformed to create another change impacts. In such situations, we move to the impact transformation step.

6.5.3 Impact Transformation

When two impacts are balanced, they may introduce another impact that is created due to the combination of the two change operations. The balancing of two or more impacts may transform existing impacts to other impacts, which are not observed at atomic change levels. For example, in case of balancing impacts, even if we remove the impacts, the operation may indicate generalization or specialization in the case of operations that alter hierarchies. Here after balancing impacts, we should check whether we are generalizing the entity by allowing it to go up in the hierarchy (generalization) or specializing the entity by allowing it to go down in the hierarchy (specialization).

The major impacts created by impact transformation are semantic impacts such as generalization, specialization, and incomparable. These impacts are created by deletion and addition of subclassof, subPropertyOf and instanceOf axioms. For example, when an instanceOf axiom is added to an instance which links it to a parent more general than its current parent and another operation deletes the instanceOf axiom of the instance from its previous parent, then we consider this as a generalization of the instance as it becomes an instance of a super class.

When two operations are candidates of balancing and if the target involves subclassOf, subPropertyOf and instanceOf axioms, then the change operations are candidates for transforming impacts.

- **Rule 4.** When impacts of two change operations balance and if the operations are applied to subsumption (subClass, subDataProperty, subObjectProperty and classAssertion axioms), the balancing impacts will transform to generalization, specialization or incomparable impacts.

  $Imp\{ACh_1, ACh_2\}$ =ES(x) if ACh_1 and ACh_2 balance and if $y \subset y'$

  $Imp\{ACh_1, ACh_2\}$ =EG(x) if ACh_1 and ACh_2 balance and if $y' \subset y$

  $Imp\{ACh_1, ACh_2\}$ =Inc(x) if ACh_1 and ACh_2 balance and $y \not\subset y' \wedge y' \not\subset y$ for ACh_1 = $AddSubclassOf(x, y)$ and ACh_2 =$DeleteSubclassOf(x, y')$

$Imp\{ACh_1, ACh_2\}$ = $Imp\{ACh_2, ACh_1\}$

To further elaborate the process of transforming the impacts, we use the following rules.

$$Transformation = \begin{cases} \text{EG, if entity moves up in the hierarchy} \\ \text{ES, if entity moves down in the hierarchy} \\ \text{EI, otherwise} \end{cases}$$

In the example in table 6.10, the second structural impact of the second change operation is removed due to impact balancing. As the class is more described with the first change operation and less described with the second change operation, it is a candidate for impact transformation. Thus, the semantic impact of the first change operation will be transformed to another impact and the transformation is determined by the current location of the target entity. In this case, the semantic impact is generalization, because the class *Deleting* goes up in the hierarchy.

Table 6.10: Impact transformation using Rule-4

No	Change Operation	Structural Impact	Semantic Impact
1	*Add SubClassOf (DeletingFile, Activity)*	*AA(FullAxiom)*	*CMD(DeletingFile)* *CMD(Activity)*
2	*Delete SubClassOf (DeletingFile, Deleting)*	*DA(FullAxiom)*	*CLD(DeletingFile)* *CLD(Deleting)*
After Transformation			
1	*Add SubClassOf (DeletingFile, Activity)*	None	*GC(DeletingFile)*
2	*Delete SubClassOf (DeletingFile, Deleting)*	None None	None *CLD(Deleting)*

Finally, all the impacts balance each other. The candidate impacts transform to generalization of the class *DeletingFile*. However, the other impacts still exist as *CMD(Activity)* and *CLD(Deleting)*. We assign the transformed impact only for the addition operation, because the addition change operation introduces the new position of the entity.

6.6 Evaluation of the Change Impact Analysis

The main research question focuses on finding an appropriate method for identification and analysis of impacts of change operations in OCMS. This includes identification of impacts of individual atomic change operations and composite change operations. In this chapter, we proposed a change impact analysis method. We build a prototype to implement the proposed method. The prototype takes an OCMS and a requested change operation and generates impacts of individual change operations and composite change operations separately. The prototype has passed through a standard unit testing. The test includes the accuracy of the analysis using test cases. However, the primary objective of the evaluation is not the unit testing of the prototype. But, using the prototype, we evaluate the accuracy of the proposed method in identifying impacts of atomic, composite and domain specific change operations. We conduct experiments to evaluate the accuracy and adequacy of the proposed solution and further compare the effectiveness of the composite change impact analysis with the individual change impact analysis.

6.6.1 Experiment Setup

To evaluate the change impact analysis method, we extend the experiment used in Section 4.6.1.1. We select and present only 10 change operations, from more than 2,000 tests cases used to evolve the respective ontologies. We analyse the impacts of the change operations and evaluate the accuracy of the results using precision. The precision calculates the number of correctly identified impacts over the number of identified impacts by the system.

6.6.2 Experimental Results

Accuracy. We evaluate the accuracy of the CIA using precision. Precision in this context measures the number of correctly identified impacts compared to the number identified impacts. The precision of the CIA is given as:

$$P(CIA) = \frac{|CIImp|}{|IImp|} \tag{6.1}$$

where:

$P(CIA)$= Precision of the change impact analysis

$|CIImp|$= Number of Correctly Identified Impacts

$|IImp|$= Number of Identified Impacts

An identified impact should satisfy the following criteria to be considered as correctly identified impact. First, the method should identify the correct impact. This means the associated impact should actually occur in the OCMS. Second, the system should identify the affected entity correctly. When both criteria are satisfied, we consider the impact as correctly identified impact.

Based on this, we present two levels of evaluation results. The first level presents the precision of the CIA using a single change operation. The change operation used is *Delete class (Activity)* from the software help management case study. In this evaluation, the analysis is conducted using attach-to-parent, cascade and no-action strategies. We present the precision of the framework in Table 6.11.

Table 6.11: Precision of impacts of a single change operation

Effect	No-action			Cascade			Attach-to-parent		
	CIImp	IImp	P(CIA)%	CIImp	IImp	P(CIA)%	CIImp	IImp	P(CIA)%
Class Less Described	15	15	100	14	14	100	3	3	100
Object Property Less Described	1	1	100	1	1	100	1	1	100
Instance Less Described	-	-	-	21	21	100	-	-	-
Class Generalized	-	-	-	-	-	-	12	12	100
Class Deleted	1	1	100	29	29	100	1	1	100
Axiom Deleted	15	15	100	60	60	100	3	2	100
Orphan Classes	12	12	100	-	-	-	-	-	-
Instances Deleted	-	-	-	7	7	100	-	-	-
Overall Precision	-	-	100	-	-	100	-	-	100

The precision result shows that the change impact analysis process identifies those impacts of the change operations over three different change implementation strategies. This precision result is based on the change impact analysis method we proposed. This means, for the impacts we defined, the change impact analysis method identifies them whenever they occur during the evolution process. For a single change operation, the result shows a 100% precision and passes our requirements. The empty rows represent impacts that are not observed in that specific strategy.

The second level presents the average precision of the CIA framework based on the analysis result of 10 frequent change scenarios taken from all the case studies. We analysed the individual change scenarios using CIA and compute the average precision for a number of change scenarios. The average precision of the CIA framework is given in Table 6.12.

Table 6.12: Average precision of impacts of multiple change operations

Change Operation	No-action	Cascade	Attach-to-parent
	P(CIA)%	P(CIA)%	P(CIA)%
Delete Class(*Student*)	100	100	100
Add DisjointClasses(*Staff*, *Student*)	100	100	100
Delete Instance(*John*)	100	100	100
Delete Class (*Table*)	100	100	100
Add SubClassOf(*Schema*, *Relation_Schema*)	100	100	100
Delete ObjectProperty(*hasSchema*)	100	100	100
Add Class (*GUI*)	100	100	100
Delete DataProperty (*hasAverageSize*)	90	90	90
Delete Instance(*id-123.xml*	100	100	100
Add Instance (*id-1234/xml*, *File*)	100	100	100
Average Precision	99	99	99

The results in Table 6.12 show the average precision of the change impact analysis (CIA) method over 10 change operations taken from the three case studies. These change operations represent frequent scenarios and are used to evolve the ontology using the three selected strategies whenever the strategies are applicable. For each change operation, we measure the precision as shown in Table 6.12. For 9 of the change operations, the result shows 100% accuracy. However, in the case of one of the change operations, the accuracy of the change operation is 90%. This is because the impact analysis identified an impact which does not occur in the OCMS.

The 90% result is attributed to the impact analysis associated to data properties. This result is observed consistently throughout the three case studies. We investigated the cause of such exceptional result. The problem arises from the existence of a false positive output. This means, the change impact analysis approach identified and reported an impact that is not actually occurring in the OCMS. We further examine the source of the false positive result. The result is an outlier which may arise from the specification used in the OWL API or the interpretation of the OWL 2 specification. Further study needs to be conducted to understand and remove the false positive impacts identified by the change impact analysis method.

The result shows that the change impact analysis gives satisfactory level of precision for implementing different change operations over different case studies. As the evaluation involves different case studies, it shows a promising result, which gives a justification for the applicability of the proposed solution in different domain areas.

6.6.3 Comparison with Existing Tool

To further validate the results of the change impact analysis process, we compare the results of our system with similar systems. The system we choose is the protege software. We evolve the three ontologies using the change operations used in Table 6.12. We used three evolution strategies to evolve the ontologies. The Protege 4.2 ontology editor allows comparison of two ontology versions and shows the additions, modifications and deletions in the new version. We present the result identified from Protege and our change impact analysis (CIA) in Table 6.13. The result shows that the change impact analysis approach identified all impacts that are identified by protege. In addition to that, our approach identified additional change impacts. There are additional structural and semantic impacts that are identified by our system. This is mostly attributed to semantic impacts which are not supported by protege ontology comparison tool.

Following the above procedure for the selected change operations, we present the overall accuracy of our system. We further present the additional structural and semantic impacts. These additional impacts actually occur in the system and explain the impacts of the change operations. Both the results show that our proposed approach provides accurate and additional information to the ontology engineer.

Table 6.13: Identified change impacts: A comparison between Protege and CIA

	Protege	CIA	Accuracy
No Action			
Structural Impacts	DC(Student)	DC(Student)	
	DA(Student subclassOf Person)	DA(Student subclassOf Person)	
	DA(UnderGraduateStudent subclassOf Student)	DA(UnderGraduateStudent subclassOf Student)	
	DA(PHDStudent subclassOf Student)	DA(PHDStudent subclassOf Student)	
	DA(MastersStudent subclassOf Student)	DA(MastersStudent subclassOf Student)	100%
		OC(UnderGraduateStudent)	
		OC(PHDStudent)	
		OC(MastersStudent)	Additional 3
Semantic Impacts		CLD(UnderGraduateStudent)	
		CLD(PHDStudent)	
		CLD(MastersStudent)	
		CLD(Person)	Additional 4
Cascade			
Structural Impacts	DC(Student)	DC(Student)	
	DC(UnderGraduateStudent)	DC(UnderGraduateStudent)	
	DC(MastersStudent)	DC(MastersStudent)	
	DC(PHDStudent)	DC(PHDStudent)	
	DA(Student subclassOf Person)	DA(Student subclassOf Person)	
	DA(UnderGraduateStudent subclassOf Student)	DA(UnderGraduateStudent subclassOf Student)	
	DA(MastersStudent subclassOf Student)	DA(MastersStudent subclassOf Student)	
	DA(PHDStudent subclassOf Student)	DA(PHDStudent subclassOf Student)	
	DI(Javed)	DI(Javed)	
	DI(Peter)	DI(Peter)	
	DI(Pooyan)	DI(Pooyan)	
	DI(Tom)	DI(Tom)	
	DI(Yalemisew)	DI(Yalemisew)	100%
		DI(Kosala)	
		DA(OPA(hasOddice Kosala,L204))	
		DA(DPA(hasFirstName Kosala “Kosala”	
		DA(DPA(hasLastName Kosala “Yapa Bandra”	
		DA(DPA(hasID Kosala “50505050”	
		DA(CA(PHDStudent, Javed))	
		DA(CA(UndergraduateStudent, Peter))	
		DA(CA(PHDStudent, Pooyan))	
		DA(CA(UndergraduateStudent, Tom))	
		DA(CA(PHDStudent, Kosala))	
		DA(CA(PHDStudent, Kosala))	
		DA(CA(MastersStudent, Janet))	
		DA(CA(MastersStudent, Mark))	Additional 13
Semantic Impacts		OPLD(hasOffice)	
		DPLD(hasId)	
		DPLD(hasName)	
		DPLD(hasFirstName)	
		ILD(Mark)	
		ILD(Janet)	
		CLD(L204)	
		CLD(Person)	Additional 8
Attach to Parent			
Structural Impacts	DC(Student)	DC(Student)	
	DA(Student subclassOf Person)	DA(Student subclassOf Person)	
	Changed(superClassOf UnderGraduateStudent)	CG(UnderGraduateStudent)	
	Changed(superClassOf PHDStudent)	CG(PHDStudent)	
	Changed(superClassOf MastersStudent)	CG(MastersStudent)	100%
Semantic Impacts		CLD(Person)	
			Additional 1

Table 6.14: Average precision of impacts of multiple change operations

Change Operation	No-action		Cascade		Attach-to-parent	
	Accuracy	Additional (Str, Sem)	Accuracy	Additional (Str, Sem)	Accuracy	Additional (Str, Sem)
Delete Class(*Student*)	100%	3,4	100%	13,8	100%	0,1
Add DisjointClasses(*Staff*, *Student*)	100%	0,2	-	-	-	-
Delete Instance(*John*)	100%	0,1	100%	0,1	100%	0,1
Delete Class (*Table*)	100%	1,7	100%	1,7	100%	1,7
Add SubClassOf(*Schema*, *Relation_Schema*)	100%	0,4	-	-	-	-
Delete ObjectProperty(*hasSchema*)	100%	0,2	100%	0,2	100%	0,2
Add Class (*GUI*)	100%	0,1	100%	0,2	-	-
Delete DataProperty (*hasAverageSize*)	100%	1,14	100%	1,14	-	-
Delete Instance(*id-123.xml*	100%	0,9	-	-	-	-
Add Instance (*id-1234/xml*, *File*)	100%	0,2	-	-	-	-
Average Accuracy	100%	-	100%	-	100%	-

6.6.4 Comparison of Individual and Composite Impact Analysis

To see how much the composite impact analysis filters the impacts we compare the number of impacts identified by individual impact analysis and composite impact analysis. Table 6.15 presents the results of the comparison for the change operations used in Table 6.12.

Table 6.15: Comparison of Individual and composite impacts

Change	No-action			Cascade			Attach-to-parent		
Operation	Indiv	Comp	%	Indiv	Comp	%	Indiv	Comp	%
1	17	12	-29.4%	65	38	-41.4%	26	6	-76.9%
2	4	4	0%	-	-	-	-	-	-
3	9	5	-44.4%	9	5	-44.4%	-	-	-
4	26	24	-7.6%	26	24	-7.6%	26	24	-7.6%
5	3	3	0%	-	-	-	-	-	-
6	7	7	0%	7	7	0%	7	7	0%
7	2	2	0%	-	-	-	5	4	-20.0%
8	29	29	0%	29	29	0%	29	29	0%
9	24	16	-33.3%	24	16	-33.3%	-	-	-
10	2	2	0%	-	-	-	5	4	-20.0%
Average	-	-	-11.49%	-	-	-21.1%	-	-	-20.7%

The above table shows that the composite change operation removes one or more impacts of the individual change operations. This shows that the composite impact analysis is essential to filter out impacts that are cancelled, balanced or transformed. The number of impacts reduced by average of 11.49% in case of the no-action strategy, 21.1% in case of the cascading strategy and 20.7% in the case of the attach strategy. These results may vary depending on the OCMS and the number of entities used in the ontology. However, the result shows that the composite change impact analysis always guarantees a less or equal number of impacts compared to the individual impact analysis. This enables the user to focus on the refined impacts of the change request. A similar comparison is conducted between the time required to complete the identification of impacts of individual change operations and composite change operations. The results show that the average time required to finish individual change impact analysis and composite change impact analysis, respectively, is 82.7ms and 146ms for no-action strategy, 75.16 and 185.83 for the cascade strategy and 72.3 and 136.6 for the attach-to-parent strategy respectively. This shows that composite change impact analysis takes twice as much time as individual change impact analysis. This is due to the additional iterations required to find cancelling, balancing and transforming impacts. In general, the time required to conduct composite impacts that contain up to 120 atomic change operations is less than 0.5 seconds which is fast enough for change impact analysis.

6.6.5 Questionnaire Results

The precision measures the accuracy of the solution. However, we need to evaluate whether the solution is adequate. This includes evaluating whether the method is capable of identifying the impacts and affected entities, the understandability and usability of impacts to address inconsistencies and invalidities. We distributed questionnaires for four users who involve on the evaluation of the prototype. We further interviewed the users based on the questionnaire results to further understand the rationale behind their responses. After the analysis of the impact of each change scenario, the users filled a questionnaire (Appendix E) related to the adequacy, transparency and usability of the CIA framework. The average responses of the users are presented in Table 6.16

Table 6.16: Users feedback on the CIA framework

Questions	average response
CIA identified all occurring impacts	4.33
CIA identified all affected entities	4.67
CIA helps me understand the impacts	4.67
CIA highlights Integrity problems	4.33
Strongly Agree= 5, Agree= 4, Slightly Agree=3 Slightly disagree= 2, disagree= 1, Strongly disagree=0	

The result shows that the users strongly agree or agree about the adequacy of the solution. In both cases the respondents agree about the occurrence of the impacts. Some of these users; however, focused on the presentation of the impacts (User interface issue) which is not the primary concern of the evaluation. The result from the questionnaire shows that the change impact analysis method identifies the impacts and the affected entities. This helps the user understand the impacts of the changes they request before they implement them permanently. Whenever there are integrity problems, the analysis highlights the problems and the change operations responsible for the violation. In general, a response from the users is encouraging.

The result in Table 6.11 and 6.12 shows that the proposed solution demonstrate a promising result, which can be used as an input for analysing impacts of change operations. The output of the analysis can be used for the selection of an optimal evolution strategy based on the number of impacts.

6.7 Summary

The change impact analysis phase performs the analysis using two stages. The individual change impact analysis process takes an atomic change operation and analyses the impacts of the individual change operations based on the preconditions defined. This phase further takes dependency analysis results to identify the dependent entities of a changing entity. We further conduct composite change impact analysis and provide information about the detailed and the summarized impacts of a change operation. This allows ontology engineers to follow their own way of implementing changes and before they implement the changes, they can run the change impact analysis to see the structural and semantic impacts of their change operations. This gives significant analysis results and flexibility for the ontology engineer specially when there are complex change operations.

The change impact analysis process, in addition to analysing impacts of change operations, allows the ontology engineer to easily pinpoint the causes of a given impact. The semantic impacts provide a wealth of information for the ontology engineer to understand what a given change operation does beyond the obvious change on the structure. This information can be exploited to search for optimized solutions that can be used to search for alternative ways of implementing the requested change using different evolution strategies. The exploitation of the information associated with the type of impact, the reasoning type and the severity of the impacts serve as an input for optimized implementation of a requested change operation. Given different evolution strategies, we can analyse the impacts of a change operation. Based on that, we can select the strategy that ensures the implementation of the requested change with a minimum impact. The implementation chapter will discuss the details of the change implementation phase.

Chapter 7

Change Optimization and Implementation

7.1 Introduction

When ontology engineers and content managers request a change, they need to know the impacts of the changes on dependent systems. They want to conduct a what-if analysis and determine how the entities are affected. Whenever they have different options of implementing a change, users tend to compare and choose the best option. For changes that have complex and multiple impacts, understanding, comparing and selecting the best option manually is error prone and time consuming. Thus, providing methods that compare different implementation options and select the optimal one are important for a better evolution.

In Section 5.4, we pointed out that a requested change operation can be implemented using different evolution strategies. These evolution strategies are different by the type and number of change operations they contain. In Chapter 6, we analysed the impacts of change operations in general. However, the selection of the best strategy requires an in-depth analysis of the nature of impacts, the statements affected, the entities added or removed and the number of change operations. The optimal strategy that meets the requirements of the user shall be selected based on these analysis results.

Depending on the change request, we present the user with different implementation options and the associated costs of evolution for each option. The cost of evolution measures the overall effort required to implement a change in a given strategy. The cost is calculated by taking the impacts, operation types, statement types and performance into account. After the cost of each evolution strategy is calculated, the user can choose the best option with the minimum cost or let the system decide the optimal solution automatically. Finally, the user compares impacts of the change request in different evolution strategies based on their associated cost and selects the optimal strategy. Once the optimal strategy is selected, the implementation can be performed using different editors and APIs that are capable of implementing atomic change operations in a user-defined fashion.

This chapter is organized as follows. Section 7.2 presents the optimization framework and introduces its components. Section 7.3 discusses the optimization criteria and how each criterion is measured separately. A formula used to measure the cost of evolution that serves as a measure for the optimal strategy selection and implementation is discussed in Section 7.4. We evaluate the proposed solution and present the evaluation results in Section 7.5. Finally, Section 7.6 presents the summary of the chapter.

7.2 Change Impact Optimization Framework

Ontology evolution often involves analysis and selection of different strategies before implementing the changes and evolving the ontology. In this section, we propose a novel approach for the selection of an optimal strategy to implement a requested change. We propose an optimization framework, which utilizes evolution strategies, severity of change impacts, deductive and incremental changes, affected statement types and the number of change operations.

The framework begins with identifying applicable implementation strategies to implement the requested change operation. Each strategy is evaluated using four criteria, which serve as an input for calculating cost of evolution. The severity value is used to evaluate the seriousness of an impact. The operation type measures the incremental change (Addition)

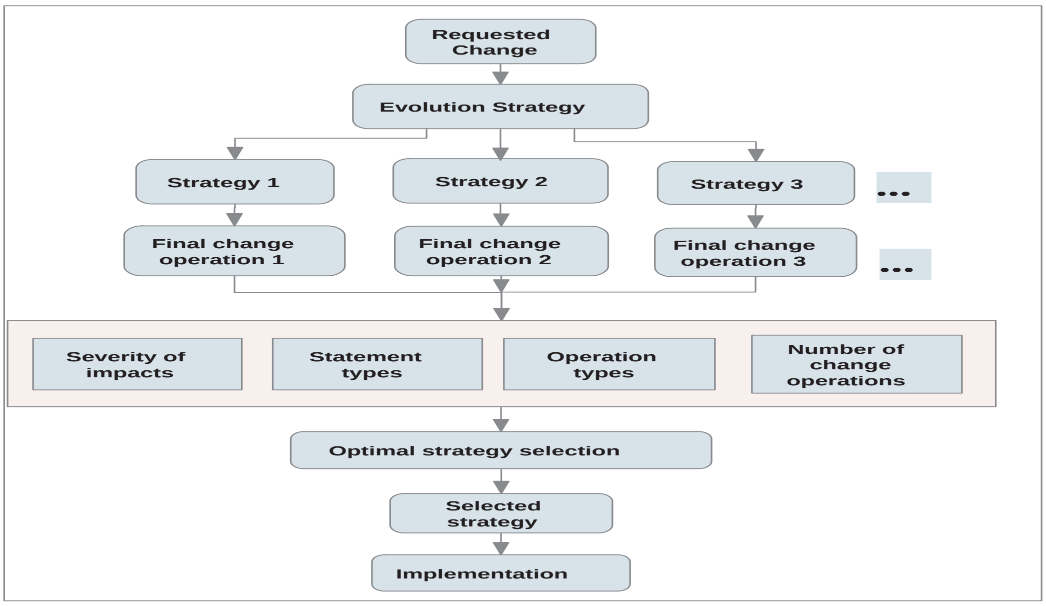


Figure 7.1: Framework for selecting optimal strategy

and decremental change (deletion). The statement type measures the number of $\mathcal{AB}$ox and $\mathcal{TB}$ox statements (6.3.3) affected. The performance measures the total number of change operations required to implement the change. The cost of evolution is measured by combining the above criteria based on their assigned weight in the evolving OCMS.

The optimal strategy selection process has the following major criteria. The first one is a severity criterion, which is responsible for calculating the severity of impacts of a given strategy. ***Severity*** measures the intensity or the degree of an impact on an OCMS in relation to the problem it causes, the effort and the level of expertise it requires to resolve the impact. The second criterion is the performance, which focuses on selecting the optimal way of executing the change in terms of the number of change operations involved. The third criterion uses the statement types ($\mathcal{AB}$ox and $\mathcal{TB}$ox) that are affected. Finally, the fourth criterion focuses on the incremental and decremental changes in terms of the number of additions and deletions. The optimal strategy selection stage estimates the overall cost of evolution using the four criteria. It further compares the cost and ranks the strategies according to their cost. The strategy with minimum cost is the strategy which is preferable for implementation. Finally, the changes in the selected strategy will be implemented.

7.3 Change Optimization Criteria

Selection of optimal implementation strategy depends on the optimization criteria set by the ontology engineer. In this research we selected four different optimization criteria, which are included in the optimization framework. These criteria are discussed in detail in the following sections.

7.3.1 Severity of Impacts

In Chapter 6, we identified structural and semantic impacts. We observed that some of these impacts are severe and cause more problems than the others. Thus, it becomes important to distinguish between the impacts based on their severity. Severity measures the degree of seriousness of a given impact. To quantify the severity of impacts, we propose a quantitative estimation on a scale of 0 to 100. A severity value 0 is assigned to impacts with minimum severity and is interpreted as an impact, which does not create any problem if it occurs in the OCMS. The value 100 refers to an impact with a high degree of impact, which makes the OCM erroneous or degrades its importance. Any value in between indicates the degree of severity of the associated impact.

Assigning an exact value for severity of an impact is not a trivial task. It requires a deep knowledge of the impact and the problems associated to the impact. When an entity is impacted, we need to know how serious the impact is, how much time it requires to address the problem, and what level of expertise it requires to understand and resolve it. To facilitate the process of estimating the severity value, we propose four levels of severity categories. The categories are low impacts (0-25) moderate impacts (26-50), high impacts (51-75) and crucial impacts (76-100). This categorization is used to roughly group impacts based on an estimated value. For example, if we have unsatisfiable class (UC) impact, first we determine whether the impact is low, moderate, high or crucial. If UC is crucial, then we determine how crucial it is and assign a value between 76 and 100. In an OCMS where UC impact is unacceptable, we assign a severity value close to 100. This approach guides the ontology engineer to group impacts using the four categories, and then assigns a severity value within the range of the category.

The severity value is not uniform across all OCMS. It is defined in relation to a given OCMS. As we discussed in Section 4.2, there are different kinds of OCMS. In each OCMS, the nature of the ontology, the annotation and the content are different. In a similar way, the objective of the OCMS is different. This difference may lead to an assignment of different severity values for a single impact in different OCMS. Thus, a very severe impact in one OCMS may not be that severe in another OCMS. For example, in one OCMS the system orphan instances are not allowed. This makes the orphan instance (OC) impact very severe in that OCMS. However, in other OCMS orphan instances are allowed. This makes the orphan instance (OC) impact less severe.

Thus, setting severity values of impacts in a given OCMS depends on the requirements defined by the ontology engineer or content manager. We use heuristics to measure the severity value of the impacts. The heuristics consider criteria such as the tolerance of a given OCMS to a given impact, the amount of time and expertise required to reduce or avoid the impact and the semantic information we lose or gain due to a given impact. In general, there are impacts of change operations that introduce errors in the system unless they are resolved. There are other impacts that cause the OCMS to introduce integrity violations in part without affecting the whole. Other impacts only cause the loss of some semantics.

A severity value is assigned by experts who are designing the OCMS. The assignment may vary according to the design specification, the purpose and other factors. For the purpose of the experiment, we calculated the average severity value of impacts from different estimations by experts. This average value is used as a default value for this experiment. However, the actual severity values assigned by the user may significantly vary from the average value depending on the target OCMS and the user's preference. When the user does not supply the values, the average will be taken as a default value. Depending on the nature of the OCMS, the preference of the ontology engineer and the content manager, the values can be configured.

Calculating the severity of the requested change operation is the last process that needs

Table 7.1: Default value for severity of impacts

No.	Semantic Impact	Acronym	Severity
1	Entity More Described	(CMD,DPMD,OPMD,IMD)	15
2	Entity Less Described	(CLD,DPLD,OPLD,ILD)	75
3	Entity More Restricted	(OPMR)	75
4	Entity Less Restricted	(OPLR)	35
5	Entity More Expanded	(AME)	60
6	Entity Less Expanded	(ALE)	80
7	Entity Generalized	(CG,DPG,OPG,IG)	50
8	Entity Specialized	(CS,DPS,OPS,IS)	70
9	Entity Incomparable	(CInc, DPInc, OPInc, IInc)	70
10	Unsatisfiable Class/Property	(UC,UDP,UOP)	100
11	Invalid Instance/ Instance Property	(II, IIP)	80
No.	**Structural Impact**	**Acronym**	**Severity**
1	Orphan Classes	(OC)	80
2	Orphan Instance	(OI)	75
3	Property Cyclic Reference	(OPCR/DPCR)	90
4	Class Cyclic Reference	(CCR)	95
5	Null Reference to Content Layer	(NRC)	70
6	Null Reference to Ontology Layer	(NRO)	70

to be done. We analyse the severity of the impacts after the composite change impact analysis is performed. Severity of impacts of change operations is assigned for each individual impact identified in Section 6.3. These observed impacts are the actual impacts that occur at the implementation phase of the change operation. We take the default severity value assigned for each structural and semantic impact in Table 7.1 and assign them to the respective impacts of a change operation. The selection of the optimal solution depends on the quality of the severity value assigned to the impacts. This means, if a representative severity value is assigned to an impact, the selection of the optimal solution will become accurate. Thus, ontology engineers need to carefully select a representative severity value for the impacts.

Severity Threshold. A given change operation may contain two or more impacts.

Among the impacts, there may be a few severe impacts, which need to be resolved or avoided at all cost. To calculate a representative measure of the severity of a strategy, we define a severity threshold. The severity threshold (T) sets a severity value which serves as a cutting point for impacts that are not allowed to occur in a given OCMS. If one or more impacts have a severity value greater than the threshold value, we take the maximum severity value as a representative value for that specific strategy [Johnson, 2011] [Trivedi, 2002] [Sacks et al., 1989]. A representative severity value (S) for a strategy is selected based on the severity of the individual impacts in the strategy. $s = \{s_1, s_2, \ldots, s_k\}$ represents the severity of the individual impacts contained in the strategy . If the individual severity value (s_i), where $i \in \{1, 2, \ldots, k\}$ is greater than the threshold (T), we select the maximum severity $MAX(s)$, otherwise we calculate an average severity value $AVG(s)$. Note that k represents the number of individual impacts of a change operation.

$$S = \begin{cases} \textit{MAX(s)} \text{ if } MAX(s) \geq T \\ \\ \textit{AVG(s)} \ otherwise \end{cases}$$

For example, if a threshold is set to be T=75, anything greater than 75 will be considered as crucial impact and will be picked as the severity value of the strategy, otherwise we calculate the average severity. If there is unsatisfiable class (UC) with a severity value of 100, and $T = 75$ then, we select the severity value of 100 as a representative value.

If the maximum severity of the individual impacts in a strategy is less than the threshold, we take the average severity value. f_i represents the frequency of s_i

$$AVG(s) = \frac{\sum_{i=1}^{k} s_i \times f_i}{\sum_{i=1}^{k} f_i}, \tag{7.1}$$

We take this approach to reduce the effects of frequent but less severe impacts on the overall estimation of impacts. By definition, crucial impacts should be avoided by any means. To ensure this we should set a threshold that serve as a pivot for severity. Anything which is less than the threshold is represented by the average severity. Let us take an example (Table 7.2) to show how the severity calculation works and compare severity of impacts of a change operation.

Table 7.2: Severity value calculation

Strategy 1		Strategy 2	
Impacts	Severity	Impacts	Severity
CLD	75	OC	100
CMD	75	CMD	75
OPLR	35	UC	100
		II	80
Average	61.6	Max	100

Let us set the severity threshold to 80. In the first strategy, since all the severity of the impacts is less than 80 we calculate the average severity as a representative value. Thus, the representative value is 61.6. However, in the second strategy, since there are OC, UC and II impacts which are greater than or equal to the threshold value, we take the maximum severity as a representative value, which is 100.

Based on the above calculation, we present the severity values of different strategies for the change operation (***Delete Class Activity***) implemented in our case study (Appendix A). This change operation has the following representative severity values for each of the applicable strategies.

Table 7.3: Severity value

	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
Severity	80.00	56.00	56.00	75.00	75.00

Let us tune the severity value to represent OCMS that gives less severity to Orphan classes and orphan instances. To do this we modify the values in Table 7.1 as follows. Severity of OC is changed from 80 to 10 and OI impact is changed from 75 to 10. This shows that the OCMS is not sensitive to the existence of orphan classes and instances. Thus, the severity value become different from the above result in Table 7.3

Table 7.4: Severity value- different value for OC and OI Impacts

	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
Severity	47.00	56.00	56.00	68.00	68.00

7.3.2 Type of Change Operation (Addition and Deletion)

Addition and deletion operations are used as criteria for selecting an optimal strategy. If the ontology evolution favours incremental evolution, which adds new knowledge every time without deleting existing knowledge, the final change operations are expected to introduce more addition operations compared to deletion operations. In this case, the removal of a given entity and the introduction of a new entity may not be considered to have the same impact. Thus, the type of the operation is considered as another factor to determine the optimal implementation strategy. The addition operation is different from deletion in the following ways. When we add a new entity, we may need to search existing entities, but the search is specific to an entity. This means, there may not be much time and resource wasted to add the new entity in the OCMS. However, when we delete an entity, first we need to conduct a dependency analysis, which includes searching all dependent entities. Second, cascade the change to all dependent entities. In terms of time and resource, a deletion operation incurs extra cost compared to addition.

Whenever there is a difference of performance between addition and deletion operations, we assign a different weight to the change operations [Trivedi, 2002] [Sacks et al., 1989]. We assign W (A) for the associated weight of addition operations and W (D) for the associated weight of deletion operations. The lesser the weight, the higher the desirability of the change operation. Higher weight indicates the less desirability of the change operation in the strategy. Thus, for a given final change operation, the weighted frequency of addition operations and deletion operations are used. This measure makes this parameter quantifiable and facilitates comparison of one strategy with another in terms of change operations.

$$WF(A) = W(A) * |A| \tag{7.2}$$

$$WF(D) = W(D) * |D| \tag{7.3}$$

$$OT = WF(A) + WF(D) \tag{7.4}$$

Where:

OT= Operation Type

WF(A) is weighted frequency of Additions

WF(D) is weighted frequency of Deletions

$0 \leq W(A) \leq 1$, $0 \leq W(D) \leq 1$ and $W(A)$+ $W(D) = 1$

$|A|$ = number of additions and $|D|$ = number of deletions

Let us look at the weighted calculation for addition and deletion operations. Let w(A) = 0.25 and W(D) = 0.75 as shown in Table 7.5.

Table 7.5: Frequency of additions and deletions

	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
Number of Additions	0	12	12	0	0
Number of Deletions	16	16	16	96	89
Weighted Frequency of Additions and Deletions					
Number of Additions	0	4.80	4.80	0.00	0.00
Number of Deletions	9.60	9.60	9.60	57.60	53.60
Operation type					
Operation Type	9.60	14.40	14.40	57.60	53.40

7.3.3 Statement Types ($\mathcal{AB}$ox and $\mathcal{TB}$ox)

In ontologies, changing the $\mathcal{TB}$ox statements may affect all the $\mathcal{AB}$ox statements associated with it. However, changing the $\mathcal{AB}$ox statements does not change the $\mathcal{TB}$ox. From all the empirical studies, we found that the $\mathcal{TB}$ox and the $\mathcal{AB}$ox statements are not equally important in different application domains and do not have equal weight. For example, in the university administration case study, it is preferable to change the $\mathcal{TB}$ox statements to amend inconsistency than the $\mathcal{AB}$ox statements. Changing the $\mathcal{AB}$ox statements means changing the information of an individual student or department. In the database case study, the classes define the individuals, thus, large weight is given to the $\mathcal{TB}$ox statements. Other applications treat both statements as equally important. This indicates that statement type serves as a means of selecting an optimal implementation strategy whenever there is a distinction on changing $\mathcal{AB}$ox statements and $\mathcal{TB}$ox statements. This criteria correspond to the OWL profiles discussed in Section 2.3.2. Ontologies adhering to OWL-QL are more sensitive to $\mathcal{AB}$ox statements and OWL2 EL are more sensitive to $\mathcal{TB}$ox statements.

Thus, the weight of the $\mathcal{AB}$ox and the $\mathcal{TB}$ox statements depend on the application and the preference of the ontology engineer. We take the weighted frequency of the strategies to measure $\mathcal{AB}$ox and $\mathcal{TB}$ox. These weighted frequencies will be used to compare final change operations in terms of statement types. The weight of $\mathcal{AB}$ox statements is given by $W(\mathcal{AB}ox)$ and the weight of $\mathcal{TB}$ox statements is given by $W(\mathcal{TB}ox)$. The weight is a value between 0 and 1.

$$WF(\mathcal{AB}ox) = W(\mathcal{AB}ox) * |\mathcal{AB}ox| \tag{7.5}$$

$$WF(\mathcal{TB}ox) = W(\mathcal{TB}ox) * |\mathcal{TB}ox| \tag{7.6}$$

$$ST = WF(\mathcal{AB}ox) + WF(\mathcal{TB}ox) \tag{7.7}$$

Where:

ST= Statement Type

$WF(\mathcal{AB}ox)$ is weighted frequency of $\mathcal{AB}ox$ statements

$WF(\mathcal{TB}ox)$ is weighted frequency of $\mathcal{TB}ox$ statements

$0 \leq W(\mathcal{AB}ox) \leq 1 \wedge 0 \leq W(\mathcal{TB}ox) \leq 1$

$|\mathcal{AB}ox|$= number of $\mathcal{AB}$ox statements, $|\mathcal{TB}ox|$= number of $\mathcal{TB}$ox statements

The number of $\mathcal{AB}$ox and $\mathcal{TB}$ox statements and the weighted frequency of $\mathcal{AB}$ox and $\mathcal{TB}$ox statements, for a given weight $W(\mathcal{AB}ox) = 0.4$ and $w(\mathcal{TB}ox) = 0.6$, is as presented in Table 7.6.

Table 7.6: Frequencies of $\mathcal{AB}$ox and $\mathcal{TB}$ox statements

	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
$\mathcal{AB}$ox statements	0.00	0.00	0.00	34.00	27.00
$\mathcal{TB}$ox statements	16.00	28.00	28.00	62.00	62.00
Weighted frequency of $\mathcal{AB}$ox and $\mathcal{TB}$ox statements					
$\mathcal{AB}$ox statements	0.00	0.00	0.00	13.60	12.80
$\mathcal{TB}$ox statements	9.80	16.80	16.80	37.20	37.20
Statement type					
Statement type	9.80	16.80	16.80	50.80	48.00

7.3.4 Performance of Change Operations

Performance measures the number of atomic change operation required to implement the change. This is calculated by counting the number of atomic change operations in the final change operation. The assumption behind this criterion is to compare the time and effort required to implement change operations especially for those content-based systems that have large number of instances or for classes with many dependencies. For each evolution strategy, we count the number of atomic change operations in the final change operation. This measure is useful when there is a need to compare strategies using number of change operations irrespective of their type or the statements they affect.

$$P = |ACh \in CCh| \tag{7.8}$$

Where:

P= Performance

ACh= Atomic change operations

CCh = Composite change operations

Table 7.7: Number of change operations

	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
Number of change operations	16	28	28	96	89

7.4 Cost of Evolution and Optimal Strategy Selection

7.4.1 Cost of Evolution

Measuring the cost of evolution to select the optimal strategy based on impact analysis is the central process of the implementation phase. We suggest two approaches to calculate the cost of evolution.

The first one is by comparing strategies using a selected individual criterion. For example, strategies can be compared using severity criterion. If there are strategies with equal severity, we can further compare them with a second criterion, say statement type and continue including the next criterion until we identify the best strategy. Our system provides analysis and selection of impacts based on a single criterion or a cascade of selected criteria. In this approach, to determine the cost of evolution, we compare the strategies by cascading the selected criteria. The user ranks the criteria with priority and we evaluate impacts using the highest priority criteria first and the lowest priority criteria last. The implementation of this approach is straightforward and it exploits the analysis results of each criterion.

The second approach uses all the criteria to measure the cost of evolution. To measure the cost of evolution, we need to evaluate all of the above criteria together. The cost of evolution becomes important as it includes all the criteria that affect the decision of the ontology engineer. All the criteria and the measures discussed above may not be equally important. An ontology engineer may assign a higher weight for the severity of impacts and ignore the number of change operations, or give more weight to the statement types and ignore additions and deletions. Thus, we need to compare each of the strategies at an individual basis to select the optimal strategy for the given criteria at hand. However, a single criterion does not characterize the requested change operation. A comprehensive measure that takes all the above criteria into account is important. To achieve this, we assign a weight to each criterion. The ontology engineer sets a weight for all criteria based on their importance in a given OCMS.

We assign weights to each criterion $\{w_1, w_2, w_3, w_4\}$ for each of the criteria chosen by the ontology engineer. Once the weights are assigned, the next stage is to calculate the cost of implementing the change operation using the given strategy. These weights are different from the previous individual weights. The weights here measure the importance of a criterion compared to the other three criteria. The individual weights measures the weights of individual criteria compared to its pair, Addition with Deletion and $\mathcal{AB}$ox with $\mathcal{TB}$ox.

$$cost(strategy) = \sum_{k=1}^{4} w_k * Cr_k \tag{7.9}$$

Where:

$Cr_k \in \{$S, ST, OT, P$\}$

$w_k \in \{w_1, w_2, w_3, w_4\}$

$w_1 + w_2 + w_3 + w_4 = 1$ and $0 \leq w_k \leq 1$

This cost is used to measure the overall impact of the change operation. Now let us look at how the weights of the criteria affect the cost estimation. To demonstrate this we will use four different scenarios. The first one is when all criteria have equal weight. The second represents a weight customized for Software Help Management OCMS (Appendix A). The third one has its weights customized for the database OCMS (Appendix B), where more weight is given to statements than to severity values. The last one is when the weight is customized for the university ontology (Appendix C), where performance and statement types are given more weight than to severity and operation type. The scenarios and the associated weights for each scenario are presented in Table 7.8.

Table 7.8: Different weights assigned for criteria

Scenario	OCMS	Severity	Operation Type	Statement Type	Performance
Scenario 1	Software help OCMS	0.25	0.25	0.25	0.25
Scenario 2	Software help OCMS	0.70	0.10	0.10	0.10
Scenario 3	Database OCMS	0.50	0.10	0.30	0.10
Scenario 4	University OCMS	0.30	0.30	0.20	0.20

Scenario 1. This scenario assigns equal weight for each criterion. This means each criterion is given a weight of 0.25. The cost estimation of the strategies using this criterion is given in Table 7.9

Scenario 2. This scenario assigns different weights for each criterion for the purpose of the experiment. These weights are assigned based on the assumption that we want to keep the semantic and structural integrity of the OCMS compared to the other criteria. The cost estimation of the strategies using this criterion is given in Table 7.10.

Table 7.9: Cost of evolution analysis - equal weight

Criteria	Weight	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
Severity	0.25	20	14	14	18.75	18.75
Operation Type	0.25	2.40	3.60	3.60	14.40	13.35
Statement Type	0.25	2.40	4.20	4.20	12.7	12.00
Performance	0.25	4.00	7.00	7.00	24.00	22.25
Cost		28.8	28.80	28.80	69.85	66.35

Table 7.10: Summary of cost of evolution - different weights

Criteria	Weight	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
Severity	0.70	56.00	39.20	39.20	52.50	52.50
Operation Type	0.10	0.96	1.44	1.44	5.76	5.34
Statement Type	0.10	0.96	1.68	1.68	5.08	4.8
Performance	0.10	1.60	2.80	2.80	9.60	8.90
Cost		59.52	45.12	45.12	72.94	71.54

Scenario 3. This scenario assigns different weights for each criterion based on the database OCMS. These weights are assigned based on the assumption that we want to keep the semantic and structural integrity of the OCMS and the statement types. The cost estimation of the strategies using this criterion is given in Table 7.11.

Table 7.11: Cost of evolution for Database systems OCMS

Criteria	Weight	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
Severity	0.50	37.50	37.50	37.50	37.50	37.50
Operation Type	0.10	0.54	0.54	0.54	0.54	0.54
Statement Type	0.30	1.62	1.62	1.62	1.62	1.62
Performance	0.10	0.90	0.90	0.90	0.90	0.90
Cost		40.55	40.55	40.55	40.55	40.55

Scenario 4. This scenario assigns different weights for each criteria based on the requirements of the university OCMS. In this OCMS severity of impacts and performance

are given more weight than the other two. The cost estimation of the strategies using this criterion is given in Table 7.12.

Table 7.12: Summary of cost of evolution for University OCMS

Criteria	Weight	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
Severity	0.30	24.00	16.80	16.80	22.50	22.50
Operation Type	0.30	0.90	1.26	1.26	4.68	3.60
Statement Type	0.20	0.96	1.68	1.68	5.08	4.8
Performance	0.20	1.60	1.60	1.60	5.20	4.00
Cost		26.5	20.62	20.62	34.78	32.02

7.4.2 Optimal Strategy Selection

The optimal strategy selection exploits the four criteria for finding the optimal implementation strategy. The cost of a strategy measures the cost of evolving a change operation using the four criteria defined. The selection of the best strategy is based on the selection of a strategy with a minimum cost.

$$BestStrategy = MIN\{Cost(Strategy_1), \ldots, Cost(Strategy_n)\} \tag{7.10}$$

Based on this, in scenario 1, there is more than one best strategy. Strategies 1, 2 and 3 have equal cost of evolution. Strategies 2 and 3 have exactly the same change operations and change impacts, thus the actual selection is between strategy 1 and Strategy 2. Scenario 2 represents a real-world selection of weights. Based on scenario 2 the best strategies are strategy 2 and 3. In this case, strategies 2 and 3 do not have any difference because even if we attach $\mathcal{TB}$ox statements only, since there is no $\mathcal{AB}$ox statement, the two strategies yield exactly the same change operations. Thus, strategy 2 is the optimal change implementation solution based on the cost of evolution and the weights assigned by the user. In scenario 3, based on the estimated weights, all the strategies yield equal weight, thus, all the strategies have an equal cost of evolution. In scenario 4, the minimum cost comes from strategy 2 and strategy 3. In this case, the two strategies yield the optimal solution. Whenever there is no $\mathcal{AB}$ox statement affected in the OCMS, strategy 2 and strategy 3 yields the same change operations thus, yields equal cost.

7.4.3 Effect of Severity Value on the Cost of Evolution

The severity value assignment plays a major role in calculating the cost of evolution. The experiment used a default severity value, which is an average value collected from experts. To demonstrate how a change in the severity value affects the cost of evolution, we take two impacts and change their value. The change of the values reflects the real-world situation where some engineers allow orphan classes and orphan instances to exist in the OCMS.

The different cases of severity assignment and the different scenarios of weights of criteria are presented in Appendix D. We identified three different severity cases where severity is assigned differently and four weighting criteria, which assign different weights for the individual criterion. Finally we presented the cost of evolution and the optimal strategy.

7.5 Evaluation of Change Impact Optimization

The change impact optimization process focuses on the selection of the best change implementation strategy based on the cost of evolution of the OCMS system. The cost of evolution includes the severity of impacts, the $\mathcal{AB}$ox and $\mathcal{TB}$ox statements, the additions and deletions, and the number of change operations involved.

7.5.1 Experimental Setup

We extend the experiment used for evaluating the previous two phases. In this phase we use the same number of change operations and OCMS. Based on the change impact analysis results, we implement change impact optimization to find the optimal change implementation strategy. We evaluate the accuracy of the system for selecting the optimal strategy based on the selected criteria.

7.5.1.1 Precision of Optimal Strategy Selection

We evaluate whether the proposed method achieves its objective by evaluating the change impact optimization process. The evaluation mainly focuses on checking whether the system identifies the optimal solution. We select the optimal solution according to parameters selected by the user of the OCMS. We use the change operations that are used in Section 6.6. For each change scenario used in the evaluation of the CIA, the prototype ranks the strategies based on their cost of evolution as first optimal, second optimal, etc. We evaluate whether the proposed change operation is the optimal solution by manually evaluating the change operations.

Table 7.13: Percentage of identifying the first three optimal solutions

Change Operation	Accuracy of Optimal strategy selection		
	First	Second	Third
Delete Class(*Student*)	√	√	√
Add DisjointClass(Staff, Student)	√	-	-
Delete Instance(*John*)	√	√	-
Delete Class (*Table*)	√	√	√
Add SubClassOf(*Schema, Relation_Schema*)	√	-	-
Delete ObjectProperty(*hasSchema*)	√	√	√
Add Class (*GUI*)	√	√	-
Delete DataProperty (*hasAverageSize*)	√	√	√
Delete Instance(*id-123.xml*	√	√	-
Add Instance (*id-1234/xml, File*)	√	√	-

Table 7.13 shows the evaluation result. The ($\surd$) mark indicates that the system identified the optimal strategy correctly and the ($-$) represents the absence of additional strategy. This means the change is implemented using the available strategy and does not have any other way of implementing the change. From the result, it is possible to conclude that the optimal strategy selection identifies the optimal strategy for all the changes to be implemented.

The graphical representation of strategies and their comparison is presented in Figure 7.2. The figure shows different scenarios taken from different domains. In all these OCMS systems, the weight assignment has a significant effect on the selection of the strategy. When all criteria are given equal weight the graph is evenly distributed. However, when the weights are assigned a different value, the graph becomes skewed to the severity value. This is due to the large value of severity as compared to the other criteria such as performance, statement types and operation types.

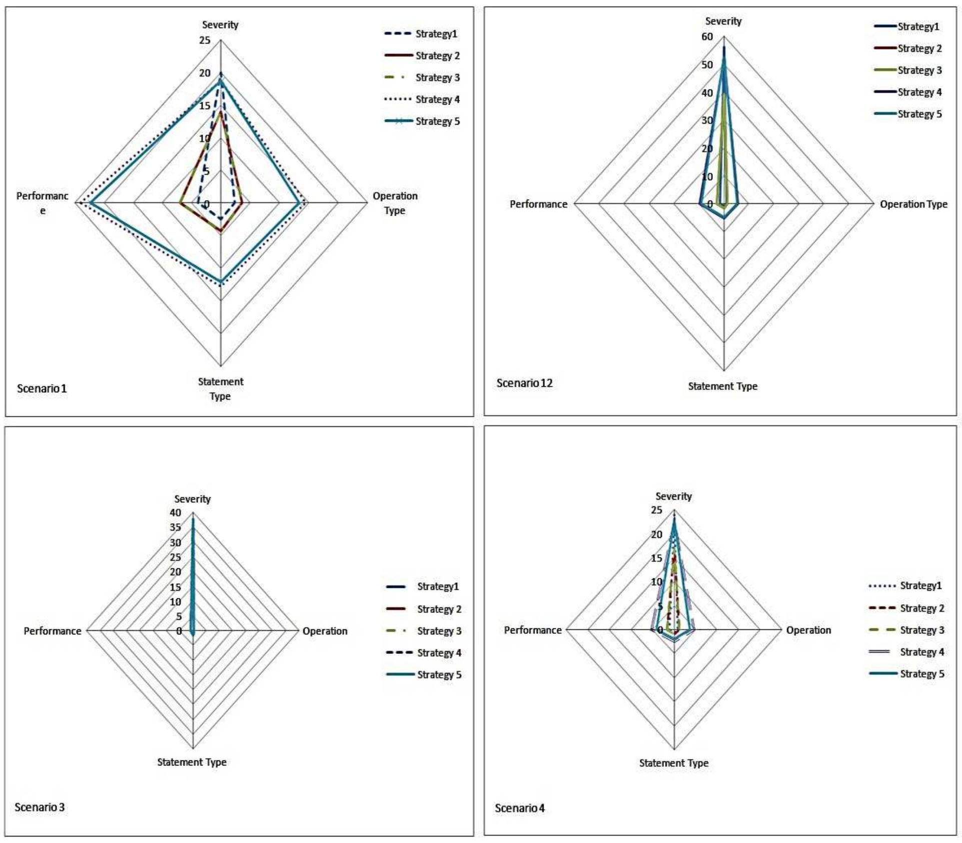


Figure 7.2: Optimal strategies for all scenarios

7.5.2 Questionnaire Results

To evaluate the usability of the CIO framework, at the end of each scenario, we distributed a questionnaire. The questionnaire aims at answering whether the change impact analysis is useful and suitable for selecting optimal strategy to evolve an OCMS.

The users evaluated the accuracy of the CIO by comparing the selected option with the other available options. If the users find the proposed option optimal, we consider the

CIO accurate and if it does not identify the optimal change implementation strategy, we consider the CIO not accurate. Table 7.13 shows the number of times the CIO identified the first, second and third optimal change operations. Table 7.14 gives a summary of the users' feedback.

Table 7.14: Users feedback on the optimal strategy selection

Questions	Average response
The cost estimation is suitable to measure impacts	4.0
I understand what I am doing at each step and understand the effects of my actions during evolution	4.0
CIO helps me find optimal strategy	3.33
Strongly Agree= 5, Agree= 4, Slightly Agree=3 Slightly disagree= 2, disagree= 1, Strongly disagree=0	

The users further provide the following feedback for the open ended questions.

- The separate presentation of the impacts of individual and composite change operations is vital to understand the impacts of the changes.

- Providing a better interface to allow users to compare all the strategies in parallel will further enhance the selection of the optimal strategy.

- The prototype needs to be customizable. This is related to setting weights of criteria and customizing the severity of the impacts.

The users agree that the optimal strategy selection is helpful to understand what is happening when a change is implemented and is useful to select the optimal strategy. Despite the effort made to avoid the bias arising from the user interface, some of the users pointed out that the presentation of the optimal strategy has affected their response. The responses for the open ended questions reinforce the need for customizability of severity of impacts and the cost of evolution to fit the requirements of the users. A comparative presentation of the alternative strategies in a single view is an important aspect. However, it relates to the user interface issue, which is not the primary objective of the prototype. In general, the system provides us with an encouraging result in relation to selecting an optimal strategy.

7.5.3 Discussion

The experimental result shows that the optimal strategy selection method is capable of selecting optimal strategies based on an individual criterion or a combination of the available criteria. The first advantage of the method is, it allows users to assign severity values to impacts and to tune the estimation towards their requirements. This means when the OCMS at hand tolerates specific impacts, the user can assign a minimum severity value (0) and when the impact is very sensitive, the user can assign a large severity value (100) for that specific impact. The second advantage of the method is its adaptability to assign weight to individual criterion by the user. This makes the method to be flexible to different environments. The evaluation result demonstrates that the optimal strategy selection is correct. The user response shows that the system helps users to understand impacts and suggest them strategies with minimum impact. However, users who participated in the evaluation further suggest improvements to be made to the presentation of the analysis results. Even if it is related to user interface issues, the users further suggest a parallel comparison between the strategies.

7.6 Summary

Change implementation is the last phase of the analysis process. In this phase we numerically analysed the impacts of change operations and provided a comparison of change operations generated using different evolution strategies. We defined different criteria for analysing and selecting the change operations that implement the change with minimum impact. The first criterion is the severity of the impacts of the change operations. For this criterion we assign a severity value for each of the impacts and use that value to compare severity of impacts among alternative change operations. The second criterion is the type of statement affected. We use $\mathcal{AB}$ox and $\mathcal{TB}$ox statements and compare strategies based on the number of $\mathcal{TB}$ox and $\mathcal{AB}$ox statements they affect. Third, we use whether the change operations introduce new class or remove existing ones. Finally, we use the performance criteria to measure the time and effort required to implement the changes. We use the number of atomic change operations to measure the performance of the available strategies of implementing the changes.

Finally, we build a method to combine all the criteria to measure the impact of change operations using inputs from the user about the weight of each of the criterion in the given OCMS. For example, in an environment where severe impacts are not allowed, even if strategies have less number of operations and few deletions, we still reject these strategies when they introduce severe impacts. In other domains, we may tend to prefer changes on $\mathcal{AB}$ox statements over $\mathcal{TB}$ox statements. Once the user sets weights for the criteria, we use the input to calculate the cost of evolution.

This approach will benefit us in the following ways. First, it allows us to select the optimal implementation strategy depending on the user's requirement. Second, it enables us to quantify impacts, required change operations and the statements they affect. Then, it permits us build an optimization technique, which is flexible and which can be customized based on the requirements of the user and the nature of the target OCMS.

In this approach, the user knows the consequences of his/her choice before the changes are implemented in the system. It quantifies the impacts of change operations and provides both qualitative and quantitative comparison of impacts. Finally, it allows the user to set the parameters flexibly and test different what-if analysis before the changes are implemented persistently.

Chapter 8

Conclusion

8.1 Introduction

This chapter presents the conclusion of the work. The chapter is organized into 3 sections. Section 8.2 discusses the major research questions and the contribution of the research in answering the major questions. Section 8.3 focuses on discussing the limitation of the research and possible future research work.

8.2 Summary of the Problem and Contribution

Ontologies are recognized as tools for enriching content serving as a source of semantics. They are used in annotations as an explicit means of embedding meaning into the content. However, due to the dynamic nature of the content and the ontologies, OCMS are subject to continuous evolution. The evolution process impacts entities and systems that are dependent on evolving entities.

The main aim of this research is the contribution of methods, tools and techniques to facilitate the evolution of OCMS systems by analysing impacts of change operations and selecting optimal evolution strategy before the changes are permanently implemented. This enables users to understand the semantic and structural impacts of the change operations on the integrity of the system by allowing a transparent, predictable and consistent evolution.

8.2.1 Contribution of the Research

This work contributes frameworks, techniques and algorithms to enhance the smooth, transparent and consistent evolution of OCMS. In this work both novel approaches and new combinations of existing research are included. This research has the following contributions.

- An OCMS framework that organizes the content, the annotation and the ontologies into separate but interdependent layers.
- A layered operator framework that represents changes based on their granularities. It facilitates the representation of domain-specific and abstract changes.
- A dependency analysis algorithm which identifies dependent entities in an OCMS. The algorithm serves as a means to identify affected entities and to generate change operations to supplement requested changes.
- A bottom-up change impact analysis approach which analyses the structural and semantic impacts of change operations.
- Algorithms that identify impacts of composite and domain-specific change operations.
- Quantitative estimation of severity of impacts which measures the seriousness of a specific impact.
- Quantitative estimation of cost of evolution of a given change operation.
- A method to select optimal implementation strategy for evolving ontologies.

The research questions and the proposed solutions are discussed as follows.

8.2.2 Capturing and Representation of Change Requests

The main research question focuses on the capturing and representation of change requests from the user. The question mainly focuses on how a requested change can be represented using executable and suitable change operations.

This research addresses the problem by proposing different solutions. The first solution is a layered operator framework which treats changes as atomic, composite and domain-specific changes. The representation further addresses the problem by separating requested changes from generated changes. The second solution is the dependency analysis algorithms. The algorithms identify dependent entities and assist the generation of change operations that are required to supplement the requested change.

8.2.3 Structural and Semantic Impact Analysis

The major question of the research is how to analyse the impacts of change operations when a given change is implemented in the system. This research question mainly focuses on analysing the structural and semantic impacts of change operations and providing the impacts of the changes on the OCMS to the user.

In relation to this question, the research identifies possible structural and semantic impacts and analyses the causes of the impacts. This phase answers the problem by exploiting the proposed layered framework of change operations. It analyses the impacts of atomic change operations first. Second, it analyses the impact of composite change operations. This phase exploits the output of the atomic change impact analysis phase to analyse impacts when two or more change operations are implemented together. The approach is flexible and is applicable to any change operation composition based on atomic change operations. This includes domain-specific changes and patterns.

8.2.4 Optimized Implementation of Changes

Once the impacts are identified, the question is how the user can select optimal implementation strategy which ensures the minimum impact. To address this problem, we use the change impact optimization approach which quantitatively measures the severity of impacts. This severity value is combined with other criteria to identify the optimal implementation strategy.

In this phase, the selection of an optimal strategy is done using individual criterion or combining different criteria. The user can use individual criterion to compare different implementation strategies. It is also possible to combine all the criteria and calculate the cost of evolution to compare different implementation strategies. One of the best features of this approach is that the users are allowed to set the severity values of the impacts relative to the OCMS at hand, assign weights to different criteria used to calculate the cost of evolution and select the optimal implementation strategy.

8.2.5 Methodology

The method used for analysis of impacts of changes in OCMS is another contribution of the study. In this research, we explored different methods to efficiently implement the evolution of ontology-based content management systems. The contributions are summarized as follows.

- The change impact analysis method applied in the context of a layered OCMS framework, layered operator framework and the change impact analysis framework is a contribution to existing research.
- The dependency analysis and the customization of evolution strategies are additional contributions of the method.
- The method further combines the change impact analysis approach with the integrity analysis.
- Finally, the method integrates optimal strategy selection and implementation of the changes. Our method allows accurate, transparent and efficient evolution of an OCMS by providing empirical evidence from three real world case studies.

8.3 Limitation and Future work

This research does not address all the problems associated with evolution of an OCMS. There are areas that are not covered in this research. We present these areas as limitations that this research does not address with the required depth.

- Sometimes ontologies use complex expressions to represent complex concepts and semantics. These complex classes are composed of two or more classes, data properties, object properties or restrictions. When a change occurs in one of the complex classes, we need to identify which specific entities are affected and which ones are not. This makes the change impact analysis process very complex. Thus, our approach does not go deep into analysing the constructs of the class expression, but we treat the whole expression as a single entity. Due to this, the research does not provide an analysis of complex expressions. However, research conducted in the area of description logic [Konev et al., 2008] [Konev et al., 2012] with different levels of expressiveness could benefit to address the limitations. Based on Description logic expressiveness, addressing complex class expression step by step could reduce the complexity of the expressions [Konev et al., 2010]. In addition to this, it is worth considering the approach used to process logical implications between the classes, individuals and properties involved in complex class expressions [Cao et al., 2006].

- This research does not cover content change to a lower level detail. We focus on the higher level changes in the content such as addition and deletion of content documents, identifiable parts of documents and their attributes. We also focus only on structured and semi-structured documents. Thus, content change which focuses on trivial textual changes is not supported.

This work addresses many of the problems identified; however, throughout the research, we discovered areas that would benefit from further investigation in the future. These areas are presented as future work in the following sections.

Change Impact Analysis

The future work in change impact analysis is to investigate detailed impacts of changes on complex class expressions. We would investigate the inconsistencies related to changes on complex class expressions. The problem with complex classes is the determination of an impact on the constituent classes of a complex class. This includes analysing which classes in the class expressions are affected and which ones are not, and how those classes are affected. This process involves decomposing the classes and further investigating the possible logical connectors and their structural and semantic importance in a given class expression.

Optimal Change Implementation

The optimal change implementation process takes four criteria to evaluate the cost of evolution of a given OCMS. The possible future work would focus on the inclusion of additional user-defined criteria. A related research direction is investigating the importance of each criterion and proposing a general formula for assigning weights based on the characteristics of the OCMS. This relieves users from specifying details of the weights of the impacts and the criteria. However, preparing a general formula requires observation from different OCMS and input from different experts in the area.

A potential future direction is the selection of change operations to avoid integrity violating change operations based on the $\mathcal{AB}$ox and $\mathcal{TB}$ox weight assigned to the OCMS. There are situations when all strategies introduce impacts with crucial severity. Then there is a possibility to select the operations to be add or remove to keep the integrity of the system. This approach enables the system to choose between operations that affect the $\mathcal{AB}$ox or $\mathcal{TB}$ox statement to avoid or reduce the observed impacts of a given strategy.

Another future direction is to investigate the selection of optimal implementation strategy using the amount of change on the inferred semantics of the OCMS. Our approach only uses asserted semantics to identify impacts. However, in the future, it is possible to estimate the cost of evolution based on the amount of change introduced in the inferred semantics. This requires the use of reasoners to infer new semantics after implementing the changes.

There are recent developments [Kondylakis & Plexousakis, 2013] in the area of ontology evolution which focuses on query rewriting, and data integration. This approach is a possible direction to follow to further enable users to analyse impacts of changes in ontologies. Such directions can be followed to address further evolution. Some of the proposed methods in belief revision [Flouris, 2006] are also worth to be considered.

Evaluation Benchmark for Change Impact Analysis One of the challenging aspects of impact analysis is the bias introduced due to the interpretation of impacts. The bias is reflected on the evaluation of change impact analysis method. An evaluation benchmark for evaluating the performance of change impact analysis tools is a potential future direction. This may include defining evaluation criteria. Currently the evaluation criteria are subjective and qualitative. To estimate the impacts of changes quantitatively, developing an all-inclusive evaluation bench mark is another future direction. The experience of the Ontology Alignment Evaluation Initiative (OAEI)[1] could serve as a starting point to introduce benchmarks for evaluating robustness of tools for ontology evolution and change impact analysis [Rosoiu et al., 2011] .

[1]http://oaei.ontologymatching.org/

Bibliography

[Abgaz & Pahl, 2012] Abgaz, Y.M.and Javed, M. & Pahl, C. (2012). Analyzing impacts of change operations in evolving ontologies. In *ISWC Workshops: Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn), 12th November, 2012, Boston, USA*.

[Abgaz et al., 2010] Abgaz, Y., Javed, M., & Pahl, C. (2010). Empirical analysis of impacts of instance-driven changes in ontologies. In *On the Move to Meaningful Internet Systems: OTM 2010 Workshops*, Lecture Notes in Computer Science.

[Abgaz et al., 2011] Abgaz, Y., Javed, M., & Pahl, C. (2011). A framework for change impact analysis of ontology-driven content-based systems. In *On the Move to Meaningful Internet Systems: OTM 2011 Workshops*, Lecture Notes in Computer Science.

[Abgaz et al., 2012] Abgaz, Y., Javed, M., & Pahl, C. (2012). Dependency analysis in ontology-driven content-based systems. In L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. Zadeh, & J. Zurada (Eds.), *Artificial Intelligence and Soft Computing*, volume 7268 of *Lecture Notes in Computer Science* (pp. 3–12).

[Adler et al., 2008] Adler, B. T., Chatterjee, K., de Alfaro, L., Faella, M., Pye, I., & Raman, V. (2008). Assigning trust to wikipedia content. In *Proceedings of the 4th International Symposium on Wikis*, WikiSym '08 (pp. 1–12). New York, NY, USA: ACM.

[Afsharchi & Far, 2006] Afsharchi, M. & Far, B. H. (2006). Automated ontology evolution in a multi-agent system. In *Proceedings of the 1st international conference on Scalable information systems*, InfoScale '06 New York, NY, USA: ACM.

[Ahmad et al., 2009] Ahmad, A., Basson, H., Deruelle, L., & Bouneffa, M. (2009). A knowledge-based framework for software evolution control. In *INFORSID* (pp. 111–126).

[Ardil, 2005] Ardil, C., Ed. (2005). *The Second World Enformatika Conference, WEC'05, February 25-27, 2005, Istanbul, Turkey, CDROM*. Enformatika, Çanakkale, Turkey.

[Arnold, 1996] Arnold, R. S. (1996). *Software Change Impact Analysis*. Los Alamitos, CA, USA: IEEE Computer Society Press.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F., Eds. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. New York, NY, USA: Cambridge University Press.

[Baader et al., 2006] Baader, F., Lutz, C., & Suntisrivaraporn, B. (2006). CEL–a polynomial-time reasoner for life science ontologies. In U. Furbach & N. Shankar (Eds.), *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence* (pp. 287–291).: Springer-Verlag.

[Baresi & Heckel, 2002] Baresi, L. & Heckel, R. (2002). Tutorial introduction to graph transformation: A software engineering perspective. In *Proceedings of the First International Conference on Graph Transformation*, ICGT '02 (pp. 402–429). London, UK, UK: Springer-Verlag.

[Bechhofer et al., 2002] Bechhofer, S., Carr, L., Goble, C. A., Kampa, S., & Miles-Board, T. (2002). The semantics of semantic annotation. In *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002* (pp. 1152–1167). London, UK, UK: Springer-Verlag.

[Bell et al., 2007] Bell, D., Qi, G., & Liu, W. (2007). Approaches to inconsistency handling in description logic-based ontologies. In *Proceedings of the 2007 OTM Confederated* *international conference on On the move to meaningful internet systems - Volume Part II*, OTM'07 (pp. 1303–1311). Berlin, Heidelberg: Springer-Verlag.

[Benjamins et al., 2002] Benjamins, V., Contreras, J., Corcho, O., & Gomez-perez, A. (2002). 'six challenges for the semantic web'. *Cristani, M(ED): KR2002 Workshop on the Semantic Web, Toulouse, France.*

[Bennett & Rajlich, 2000] Bennett, K. H. & Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00 (pp. 73–87). New York, NY, USA: ACM.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American*, 284(5), 34–43.

[Bizer & Schultz, 2008] Bizer, C. & Schultz, A. (2008). Benchmarking the performance of storage systems that expose sparql endpoints. In *Proceedings of the ISWC Workshop on Scalable Semantic Web Knowledgebase*.

[Bloehdorn et al., 2006] Bloehdorn, S., Haase, P., Sure, Y., & Voelker, J. (2006). *Ontology Evolution*, (pp. 51–70). John Wiley and Sons, Ltd.

[Bohner, 2002] Bohner, S. (2002). Extending software change impact analysis into cots components. In *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE* (pp. 175 – 182).

[Bönström et al., 2003] Bönström, V., Hinze, A., & Schweppe, H. (2003). Storing rdf as a graph. In *Proceedings of the First Conference on Latin American Web Congress*, LA-WEB '03 (pp. 27–36). Washington, DC, USA: IEEE Computer Society.

[Bounif & Pottinger, 2006] Bounif, H. & Pottinger, R. (2006). Schema repository for database schema evolution. In *Proceedings of the 17th International Conference on Database and Expert Systems Applications* (pp. 647–651). Washington, DC, USA: IEEE Computer Society.

[Boyce & Pahl, 2007] Boyce, S. & Pahl, C. (2007). The development of subject domain ontologies for educational technology systems. *Journal of Educational Technology and Society (ETS) IEEE*, 10(3), 275–288.

[Breech et al., 2005] Breech, B., Tegtmeyer, M., & Pollock, L. (2005). A comparison of online and dynamic impact analysis algorithms. In *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, CSMR '05 (pp. 143–152). Washington, DC, USA: IEEE Computer Society.

[Buckley et al., 2005] Buckley, J., Mens, T., Zenger, M., Rashid, A., & Kniesel, G. (2005). Towards a taxonomy of software change: Research articles. *J. Softw. Maint. Evol.*, 17(5), 309–332.

[Cao et al., 2006] Cao, C., Sui, Y., & Sun, Y. (2006). Logical connections of statements in ontologies. In Y. Yao, Z. Shi, Y. Wang, & W. Kinsner (Eds.), *Proceedings of the Fifth IEEE International Conference on Cognitive Informatics, ICCI 2006, July 17-19, Beijing, China* (pp. 440–446).: IEEE.

[Castagna, 1995] Castagna, G. (1995). Covariance and contravariance: conflict without a cause. *ACM Transactions on Programming Languages and Systems*, 17(3), 431–447.

[Castano et al., 2006] Castano, S., Ferrara, A., & Hess, G. N. (2006). Discovery-driven ontology evolution. In G. Tummarello, P. Bouquet, & O. Signore (Eds.), *Semantic Web Applications and Perspectives*, volume 201 of *CEUR Workshop Proceedings*: CEUR-WS.org.

[Ceravolo et al., 2008] Ceravolo, P., Damiani, E., & Leida, M. (2008). Ontology robustness in evolution. In R. Meersman, Z. Tari, & P. Herrero (Eds.), *On the Move to Meaningful Internet Systems: OTM 2008 Workshops*, volume 5333 of *Lecture Notes in Computer Science* (pp. 1010–1017). Springer.

[Ceravolo et al., 2007] Ceravolo, P., Damiani, E., & Viviani, M. (2007). Bottom-up extraction and trust-based refinement of ontology metadata. *IEEE Transactions on Knowledge and Data Engineering*, 19(2), 149 –163.

[Chu et al., 2009] Chu, H.-C., Chen, M.-Y., & Chen, Y.-M. (2009). A semantic-based approach to content abstraction and annotation for content management. *Expert Syst. Appl.*, 36(2), 2360–2376.

[Cimiano & Völker, 2005] Cimiano, P. & Völker, J. (2005). Text2onto - a framework for ontology learning and data-driven change discovery. In E. M. Andres Montoyo, Rafael Munoz (Ed.), *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, volume 3513 of *Lecture Notes in Computer Science* (pp. 227–238). Alicante, Spain: Springer.

[Cormen et al., 2001] Cormen, T. H., Stein, C., Rivest, R. L., & Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

[Cox et al., 2001] Cox, L., Harry, D., Skipper, D., & Delugach, H. S. (2001). Dependency analysis using conceptual graphs. In *Proceedings of the 9th International Conference on Conceptual Structures, ICCS 2001*: Springer.

[Şah & Wade, 2010] Şah, M. & Wade, V. (2010). Automatic metadata extraction from multilingual enterprise content. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10 (pp. 1665–1668). New York, NY, USA: ACM.

[Curino et al., 2008] Curino, C. A., Tanca, L., Moon, H. J., & Zaniolo, C. (2008). Schema evolution in wikipedia: Toward a web information system benchmark. In *International Conference on Enterprise Information Systems*.

[Davies et al., 2003] Davies, J., Fensel, D., & Harmelen, F. v., Eds. (2003). *Towards the Semantic Web: Ontology-driven Knowledge Management*. New York, NY, USA: John Wiley & Sons, Inc.

[De Leenheer & Meersman, 2007] De Leenheer, P. & Meersman, R. (2007). Towards community-based evolution of knowledge-intensive systems. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I*, OTM'07 (pp. 989–1006). Berlin, Heidelberg: Springer-Verlag.

[Dentler et al., 2011] Dentler, K., Cornet, R., ten Teije, A., & de Keizer, N. (2011). Comparison of reasoners for large ontologies in the owl 2 el profile. *Semantic Web*, 2(2), 71–87.

[Djedidi & Aufaure, 2010a] Djedidi, R. & Aufaure, M.-A. (2010a). Onto-evoal an ontology evolution approach guided by pattern modeling and quality evaluation. In *Proceedings of the 6th international conference on Foundations of Information and Knowledge Systems*, FoIKS'10 (pp. 286–305). Berlin, Heidelberg: Springer-Verlag.

[Djedidi & Aufaure, 2010b] Djedidi, R. & Aufaure, M.-A. (2010b). *Ontology Evolution: State of the Art and Future Directions*, (pp. 179–207). Ontology Theory, Management and Design: Advanced Tools and Models. IGI Global. ID: 42890.

[Edmunds & Morris, 2000] Edmunds, A. & Morris, A. (2000). The problem of information overload in business organisations: a review of the literature. *International Journal of Information Management*, 20(1), 17 – 28.

[Elmasri & Navathe, 2010] Elmasri, R. & Navathe, S. (2010). *Fundamentals of Database Systems*. USA: Addison-Wesley Publishing Company, 6th edition.

[Enkhsaikhan et al., 2007] Enkhsaikhan, M., Wong, W., Liu, W., & Reynolds, M. (2007). Measuring data-driven ontology changes using text mining. In *AusDM* (pp. 39–46).

[Eppler & Mengis, 2004] Eppler, M. J. & Mengis, J. (2004). The concept of information overload: A review of literature from organization science, accounting, marketing, mis, and related disciplines. *The Information Society*, 20(5), 325–344.

[Fensel et al., 2001] Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D., & Patel-Schneider, P. (2001). Oil: an ontology infrastructure for the semantic web. *Intelligent Systems, IEEE*, 16(2), 38 – 45.

[Fernández et al., 2011] Fernández, M., Cantador, I., López, V., Vallet, D., Castells, P., & Motta, E. (2011). Semantically enhanced information retrieval: An ontology-based approach. *Web Semant.*, 9(4), 434–452.

[Flouris, 2006] Flouris, G. (2006). On belief change in ontology evolution: Thesis. *AI Communication*, 19(4), 395–397.

[Flouris et al., 2008] Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., & Antoniou, G. (2008). Ontology change: Classification and survey. *Knowledge Engineering Review*, 23(2), 117–152.

[Flouris & Plexousakis, 2005] Flouris, G. & Plexousakis, D. (2005). Handling ontology change: Survey and proposal for a future research direction. *Artificial Intelligence*, (September), 1–55.

[Flouris et al., 2006] Flouris, G., Plexousakis, D., & Antoniou, G. (2006). A classification of ontology change. *Poster Proceedings of the 3rd Italian Semantic Web Workshop, Semantic Web Applications and Perspectives(SWAP-2006)*.

[Glimm et al., 2010] Glimm, B., Horrocks, I., Motik, B., & Stoilos, G. (2010). Optimising Ontology Classification. In P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, & B. Glimm (Eds.), *Proceeding of the 9th International Semantic Web Conference (ISWC 2010)*, volume 6496 of *LNCS* (pp. 225–240). Shanghai, China: Springer.

[Gomez-Perez & Corcho, 2002] Gomez-Perez, A. & Corcho, O. (2002). Ontology languages for the semantic web. *Intelligent Systems, IEEE*, 17(1), 54 – 60.

[Gómez-Pérez et al., 2007] Gómez-Pérez, A., Fernández-López, M., & Corcho, O. (2007). *Ontological Engineering: with examples from the areas of Knowledge Management, e-*

Commerce and the Semantic Web. (Advanced Information and Knowledge Processing). Secaucus, NJ, USA: Springer-Verlag New York, Inc.

[Goncalves et al., 2011] Goncalves, R. S., Parsia, B., & Sattler, U. (2011). Analysing the evolution of the nci thesaurus. In *Proceedings of the 2011 24th International Symposium on Computer-Based Medical Systems*, CBMS '11 (pp. 1–6). Washington, DC, USA: IEEE Computer Society.

[Grau et al., 2008] Grau, B. C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., & Sattler, U. (2008). Owl 2: The next step for owl. *Web Semant.*, 6(4), 309–322.

[Gross et al., 2009] Gross, A., Hartung, M., Kirsten, T., & Rahm, E. (2009). Estimating the quality of ontology-based annotations by considering evolutionary changes. In *Proceedings of the 6th International Workshop on Data Integration in the Life Sciences*, DILS '09 (pp. 71–87). Berlin, Heidelberg: Springer-Verlag.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 199–220.

[Gruhn et al., 1995] Gruhn, V., Pahl, C., & Wever, M. (1995). Data model evolution as basis of business process management. In *Proceedings of the 14th International Conference on Object-Oriented and Entity-Relationship Modelling*, OOER '95 (pp. 270–281). London, UK, UK: Springer-Verlag.

[Guarino, 1998] Guarino, N. (1998). *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1st edition.

[Haarslev et al.,] Haarslev, V., Hidde, K., Möller, R., & Wessel, M. The RacerPro Knowledge Representation and Reasoning System. *Semantic Web*.

[Haase & Stojanovic, 2005] Haase, P. & Stojanovic, L. (2005). Consistent evolution of OWL ontologies. In A. Gómez-Pérez & J. Euzenat (Eds.), *Proceedings of the Sec-*

ond European Semantic Web Conference, volume 3532 (pp. 182–197). Heraklion, Crete, Greece: Springer.

[Haase et al., 2005] Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., & Sure, Y. (2005). A framework for handling inconsistency in changing ontologies. In *Proceedings of the 4th international conference on The Semantic Web*, ISWC'05 (pp. 353–367). Berlin, Heidelberg: Springer-Verlag.

[Hartung et al., 2012] Hartung, M., Gross, A., & Rahm, E. (2012). CODEX: Exploration of semantic changes between ontology versions. *Bioinformatics*, 26(6), 895–896.

[Hartung et al., 2011] Hartung, M., Terwilliger, J. F., & Rahm, E. (2011). Recent advances in schema and ontology evolution. In *Schema Matching and Mapping* (pp. 149–190).

[Hassan et al., 2010] Hassan, M. O., Deruelle, L., & Basson, H. (2010). A knowledge-based system for change impact analysis on software architecture. In *Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on* (pp. 545 –556).

[Heckel, 2006] Heckel, R. (2006). Graph transformation in a nutshell. *Electronic Notes in Theoretical Computer Science*, 148(1), 187–198.

[Holohan et al., 2006] Holohan, E., Melia, M., McMullen, D., & Pahl, C. (2006). The generation of e-learning exercise problems from subject ontologies. *Advanced Learning Technologies, 2006. Sixth International Conference on*, (pp. 967–969).

[Horridge & Bechhofer, 2011] Horridge, M. & Bechhofer, S. (2011). The owl api: A java api for owl ontologies. *Semantic web*, 2, 11–21.

[Horridge et al., 2006] Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., & Wang, H. (2006). The manchester owl syntax. In *OWLED2006 Second Workshop on OWL Experiences and Directions* Athens, GA, USA.

[Horrocks, 2003] Horrocks, I., P.-S. P. (2003). Reducing owl entailment to description logic satisfiability. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2870, 17–29.

[Javed et al., 2009] Javed, M., Abgaz, Y., & Pahl, C. (2009). A pattern-based framework of change operators for ontology evolution. In *On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, volume 5872 of *Lecture Notes in Computer Science* (pp. 544–553).

[Javed et al., 2010] Javed, M., Abgaz, Y., & Pahl, C. (2010). Ontology-based domain modelling for consistent content change management. In *International Conference on Ontological and Semantic Engineering (ICOSE)*.

[Javed et al., 2011a] Javed, M., Abgaz, Y., & Pahl, C. (2011a). Graph-based discovery of ontology change patterns. In *ISWC Workshops: Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn), 24th October, 2011, Bonn, Germany.*

[Javed et al., 2011b] Javed, M., Abgaz, Y., & Pahl, C. (2011b). A layered framework for pattern-based ontology evolution. In *3rd International Workshop Ontology-Driven Information System Engineering (ODISE), London, UK*.

[Javed et al., 2012] Javed, M., Abgaz, Y., & Pahl, C. (2012). Composite ontology change operators and their customizable evolution strategies. In *ISWC Workshops: Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn), 12th November, 2012, Boston, USA.*

[Javed et al., 2011c] Javed, M., Abgaz, Y. M., & Pahl, C. (2011c). Towards implicit knowledge discovery from ontology change log data. In *knowledge Science, Engineering and Management* (pp. 136–147).

[Johnson, 2011] Johnson, J. L. (2011). *Probability and Statistics for Computer Science*. Hoboken: John Wiley & Sons.

[Jones et al., 2011] Jones, D., Oconnor, A., Abgaz, Y., & Lewis, D. (2011). A semantic model for integrated content management, localisation and language technology processing. In *2nd Workshop on the Multilingual Semantic Web (MSW2011)*.

[Jun-feng et al., 2005] Jun-feng, S., Wei-ming, Z., Wei-dong, X., Guo-hui, L., & Zhen-ning, X. (2005). Ontology-based information retrieval model for the semantic web. In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, EEE '05 (pp. 152–155). Washington, DC, USA: IEEE Computer Society.

[Jurisica et al., 1999] Jurisica, I., Mylopoulos, J., & Yu, E. (1999). Using Ontologies for Knowledge Management: An Information Systems Perspective. Paper presented at the Annual Conference of the American Society for Information Sciences. In *Proceedings of the 62nd Annual Meeting of the American Society for Information Science (ASISŠ99), Oct. 31 - Nov*, volume 4 (pp. 482–496).

[Kalyanpur et al., 2011] Kalyanpur, A., Parsia, B., Sirin, E., Grau, B. C., & Hendler, J. (2011). Swoop: A web ontology editing browser. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(2).

[Khattak et al., 2010] Khattak, A., Pervez, Z., Lee, S., & Lee, Y.-K. (2010). After effects of ontology evolution. In *Future Information Technology (FutureTech), 2010 5th International Conference on* (pp. 1 –6).

[Kiryakov et al., 2004] Kiryakov, A., Popov, B., Terziev, I., Manov, D., & Ognyanoff, D. (2004). Semantic annotation, indexing, and retrieval. *Web Semantic*, 2, 49–79.

[Klein, 2004] Klein, M. (2004). *Change Management for Distributed Ontologies*. PhD thesis, Vrije Universiteit Amsterdam.

[Klein & Fensel, 2001] Klein, M. & Fensel, D. (2001). Ontology versioning on the semantic web. In *Proceedign of 1st International Semantic Web Working Symposium.* (pp. 75–91). Stanford University, CA, USA.

[Klein et al., 2002] Klein, M., Fensel, D., Kiryakov, A., & Ognyanov, D. (2002). Ontology versioning and change detection on the web. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)* (pp. 197–212).

[Knublauch et al., 2004] Knublauch, H., Fergerson, R., Noy, N., & Musen, M. (2004). The protg owl plugin: An open development environment for semantic web applications. In S. McIlraith, D. Plexousakis, & F. van Harmelen (Eds.), *The Semantic Web ISWC 2004*, volume 3298 of *Lecture Notes in Computer Science* (pp. 229–243). Springer Berlin / Heidelberg.

[Kondylakis & Plexousakis, 2013] Kondylakis, H. & Plexousakis, D. (2013). Ontology evolution without tears. *Web Semantics: Science, Services and Agents on the World Wide Web*, (in press).

[Konev et al., 2012] Konev, B., Ludwig, M., Walther, D., & Wolter, F. (2012). The logical difference for the lightweight description logic el. *Journal of Artificial Intelligence Research (JAIR)*, 44, 633–708.

[Konev et al., 2010] Konev, B., Lutz, C., Ponomaryov, D., & Wolter, F. (2010). Decomposing description logic ontologies. In *Knowledge Representation*.

[Konev et al., 2008] Konev, B., Walther, D., & Wolter, F. (2008). The logical difference problem for description logic terminologies. In *Proceedings of the 4th international joint conference on Automated Reasoning*, IJCAR ’08 (pp. 259–274). Berlin, Heidelberg: Springer-Verlag.

[Konstantinidis et al., 2008] Konstantinidis, G., Flouris, G., Antoniou, G., & Christophides, V. (2008). A formal approach for rdf/s ontology evolution. In *Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence* (pp. 70–74). Amsterdam, The Netherlands, The Netherlands: IOS Press.

[Krotzsch et al., 2011] Krotzsch, M., Vrandecic, D., Volkel, M., Haller, H., & Studer, R. (2011). Semantic wikipedia. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(4).

[Krötzsch et al., 2007] Krötzsch, M., Vrandečić, D., Völkel, M., Haller, H., & Studer, R. (2007). Semantic wikipedia. *Web Semantics*, 5(4), 251–261.

[Kruk & McDaniel, 2009] Kruk, S. R. & McDaniel, B., Eds. (2009). Springer.

[Lee et al., 2000] Lee, M., Offutt, A. J., & Alexander, R. T. (2000). Algorithmic analysis of the impacts of changes to object-oriented software. In *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*, TOOLS '00 (pp. 61–70). Washington, DC, USA: IEEE Computer Society.

[Lee et al., 2007] Lee, S., Seo, W., Kang, D., Kim, K., & Lee, J. Y. (2007). A framework for supporting bottom-up ontology evolution for discovery and description of grid services. *Expert Systems with Applications*, 32(2), 376 – 385.

[Leenheer & Mens, 2008] Leenheer, P. D. & Mens, T. (2008). Ontology evolution: State of the art and future directions. In M. Hepp, P. D. Leenheer, A. de Moor, & Y. Sure (Eds.), *Ontology Management for the Semantic Web, Semantic Web Services, and Business Applications*. Springer.

[Lehman et al., 1997] Lehman, M., Ramil, J., Wernick, P., Perry, D., & Turski, W. (1997). Metrics and laws of software evolution-the nineties view. In *Software Metrics Symposium, 1997. Proceedings., Fourth International* (pp. 20 –32).

[Liang et al., 2006] Liang, Y., Alani, H., Dupplaw, D., & Shadbolt, N. (2006). An approach to cope with ontology changes for ontology-based applications. *Proceedings Second Advanced Knowledge Technologies DTA Symposium*, (pp. 1–8).

[Maedche et al., 2003] Maedche, A., Motik, B., Stojanovic, L., Studer, R., & Volz, R. (2003). Ontologies for enterprise knowledge management. *Intelligent Systems, IEEE.*, 18(2), 22–33.

[Maynard, 2008] Maynard, D. (2008). Benchmarking textual annotation tools for the semantic web. In B. M. J. M. J. O. S. P. D. T. Nicoletta Calzolari (Conference Chair), Khalid Choukri (Ed.), *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)* Marrakech, Morocco: European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.

[Mcguinness et al., 2002] Mcguinness, D., Fikes, R., Hendler, J., & Stein, L. (2002). Daml+oil: an ontology language for the semantic web. *Intelligent Systems, IEEE*, 17(5), 72 – 80.

[Mens et al., 2002] Mens, T., Buckley, J., Rashid, A., & Zenger, M. (2002). Towards a taxonomy of software evolution. In *Workshop on Unanticipated Software Evolution*.

[Mens & Klein, 2012] Mens, T. & Klein, J. (2012). Evolving software. *ERCIM News*, 88.

[Mika, 2007] Mika, P. (2007). Ontologies are us: A unified model of social networks and semantics. *Web Semantics*, 5, 5–15.

[Motik et al., 2007] Motik, B., Shearer, R., & Horrocks, I. (2007). A Hypertableau Calculus for SHIQ. In D. Calvanese, E. Franconi, V. Haarslev, D. Lembo, B. Motik, S. Tessaris, & A.-Y. Turhan (Eds.), *Proceeding of the 20th International Workshop on Description Logics (DL 2007)* (pp. 419–426). Brixen/Bressanone, Italy: Bozen/Bolzano University Press.

[Navigli & Velardi, 2003] Navigli, R. & Velardi, P. (2003). An analysis of ontology-based query expansion strategies. In *Proceedings of Workshop on Adaptive Text Extraction and Mining (ATEM) in the* 14^{th} *European Conference on Machine Learning (ECML)* (pp. 42–49). Cavtat-Dubrovnik, Croatia.

[Noy et al., 2006] Noy, N. F., Chugh, A., Liu, W., & Musen, M. A. (2006). A framework for ontology evolution in collaborative environments. In *5th International Semantic Web Conference* (pp. 544–558).: Springer-LNCS.

[Noy & Klein, 2004] Noy, N. F. & Klein, M. (2004). Ontology evolution: Not the same as schema evolution. *Knowledge and Information Systems.*, 6(4), 328–440.

[Noy & Musen, 2002] Noy, N. F. & Musen, M. A. (2002). Promptdiff: A fixed-point algorithm for comparing ontology versions. In *AAAI/IAAI'2002* (pp. 744–750).

[Oliver et al., 1999] Oliver, D. E., Shahar, Y., Shortliffe, E. H., & Musen, M. A. (1999). Representation of change in controlled medical terminologies. *Artificial Intelligence in Medicine.*, 15(1), 53–76.

[Oren et al., 2006] Oren, E., Mller, K., Scerri, S., Handschuh, S., & Sintek, M. (2006). What are semantic annotations. *Artificial Intelligence*, 8.

[Orso et al., 2004] Orso, A., Apiwattanapong, T., Law, J., Rothermel, G., & Harrold, M. J. (2004). An empirical comparison of dynamic impact analysis algorithms. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04 (pp. 491–500). Washington, DC, USA: IEEE Computer Society.

[Pahl et al., 2007] Pahl, C., Giesecke, S., & Hasselbring, W. (2007). An ontology-based approach for modelling architectural styles. In F. Oquendo & F. Oquendo (Eds.), *ECSA*, volume 4758 of *Lecture Notes in Computer Science* (pp. 60–75).: Springer.

[Pahl et al., 2010] Pahl, C., Javed, M., & Abgaz, Y. (2010). Utilising ontology-based modelling for learning content management. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010* (pp. 1274–1279). Toronto, Canada: AACE.

[Paralic & Kostial, 2003] Paralic, J. & Kostial, I. (2003). Ontology-based information retrieval. In *Proc. of the 14th International Conference on Information and Intelligent systems, IIS 2003* (pp. 23–28).

[Petasis et al., 2009] Petasis, G., Karkaletsis, V., Krithara, A., Paliouras, G., & Spyropoulos, C. D. (2009). Semi-automated Ontology Oearning: the BOEMIE Approach. In

Proceedings of the First ESWC Workshop on Inductive Reasoning and Machine Learning on the Semantic Web.

[Plessers & De Troyer, 2006] Plessers, P. & De Troyer, O. (2006). Resolving inconsistencies in evolving ontologies. In *Proceedings of the 3rd European conference on The Semantic Web: research and applications*, ESWC'06 (pp. 200–214). Berlin, Heidelberg: Springer-Verlag.

[Plessers et al., 2007] Plessers, P., De Troyer, O., & Casteleyn, S. (2007). Understanding ontology evolution: A change detection approach. *Web Semant.*, 5(1), 39–49.

[Qin & Atluri, 2009] Qin, L. & Atluri, V. (2009). Evaluating the validity of data instances against ontology evolution over the semantic web. *Information and Software Technology.*, 51(1), 83–97.

[Redmond & Noy, 2011] Redmond, T. & Noy, N. (2011). Computing the changes between ontologies. In *Workshop on Knowledge Evolution and Ontology Dynamics, ISWC 2011*.

[Redmond et al., 2008] Redmond, T., Smith, M., Drummond, N., & Tudorache, T. (2008). Managing change: An ontology version control system. In *In OWL: Experiences and Directions, 5th Intl. Workshop, OWLED 2008*.

[Reeve & Han, 2005] Reeve, L. & Han, H. (2005). Survey of semantic annotation platforms. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing* (pp. 1634–1638).

[Ren et al., 2004] Ren, X., Shah, F., Tip, F., Ryder, B. G., & Chesley, O. (2004). Chianti: a tool for change impact analysis of java programs. *SIGPLAN Notice*, 39(10), 432–448.

[Roddick, 1995] Roddick, J. F. (1995). A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7), 383–393.

[Rosoiu et al., 2011] Rosoiu, M.-E., dos Santos, C. T., & Euzenat, J. (2011). Ontology matching benchmarks: generation and evaluation. In P. Shvaiko, J. Euzenat, T. Heath, C.

Quix, M. Mao, & I. F. Cruz (Eds.), *Ontology Matching*, volume 814 of *CEUR Workshop Proceedings*: CEUR-WS.org.

[Ruiz et al., 2009] Ruiz, E. J., Grau, B. C., Horrocks, I., & Berlanga, R. (2009). Building ontologies collaboratively using contentcvs. In *Proceedings of the 22nd International Workshop on Description Logics (DL 2009)*.

[Ruiz et al., 2011] Ruiz, E. J., Grau, B. C., Horrocks, I., & Berlanga, R. (2011). Supporting concurrent ontology development: Framework, algorithms and tool. *Data Knowledge Engineering*, 70(1), 146–164.

[Sacks et al., 1989] Sacks, J., Welch, W., Mitchell, T., & Wynn, H. (1989). Design and Analysis of Computer Experiments. *Statistical science*, 4(4), 409–423.

[Schmidt-Schaubß& Smolka, 1991] Schmidt-Schaubß, M. & Smolka, G. (1991). Attributive concept descriptions with complements. *Artif. Intell.*, 48(1), 1–26.

[Shadbolt et al., 2006] Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The semantic web revisited. *IEEE Intelligent Systems*, 21, 96–101.

[Sherriff & Williams, 2008] Sherriff, M. & Williams, L. (2008). Empirical software change impact analysis using singular value decomposition. In *Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation* (pp. 268–277). Washington, DC, USA: IEEE Computer Society.

[Sirin et al., 2007] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics*, 5(2), 51–53.

[Stojanovic, 2004] Stojanovic, L. (2004). *Methods and tools for ontology evolution.* PhD thesis, University of Karlsruhe.

[Stojanovic et al., 2002a] Stojanovic, L., Maedche, A., Motik, B., & Stojanovic, N. (2002a). User-driven ontology evolution management. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, EKAW '02 (pp. 285–300). London, UK: Springer-Verlag.

[Stojanovic et al., 2003] Stojanovic, L., Maedche, A., Stojanovic, N., & Studer, R. (2003). Ontology evolution as reconfiguration-design problem solving. In *Proceedings of the 2nd international conference on Knowledge capture*, K-CAP ’03 (pp. 162–171). New York, NY, USA: ACM.

[Stojanovic & Motik, 2002] Stojanovic, L. & Motik, B. (2002). Ontology evolution within ontology editors. *Proceedings of the OntoWebSIG3 Workshop*, 68, 568–580.

[Stojanovic et al., 2002b] Stojanovic, L., Stojanovic, N., & Handschuh, S. (2002b). Evolution of the metadata in the ontology-based knowledge management systems. In *Proceedings of the 1st German Workshop on on Experience Management: Sharing Experiences about the Sharing of Experience* (pp. 65–77).

[Taye, 2010] Taye, M. M. (2010). The state of the art: Ontology web-based languages: Xml based. *Computing Research Repository*, abs/1006.4563.

[Thomas et al., 2010] Thomas, E., Pan, J. Z., & Ren, Y. (2010). TrOWL: Tractable OWL 2 Reasoning Infrastructure. In *the Proceeding of the Extended Semantic Web Conference (ESWC2010)*.

[Trinkunas & Vasilecas, 2007] Trinkunas, J. & Vasilecas, O. (2007). A graph oriented model for ontology transformation into conceptual data model. *Technology*, 36(1), 126–132.

[Trivedi, 2002] Trivedi, K. S. (2002). *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Chichester, UK: John Wiley and Sons Ltd., 2nd edition edition.

[Tsarkov & Horrocks, 2006] Tsarkov, D. & Horrocks, I. (2006). Fact++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence* (pp. 292–297).: Springer.

[Tudorache et al., 2008] Tudorache, T., Noy, N. F., Tu, S., & Musen, M. A. (2008). Supporting collaborative ontology development in protege. In *Proceedings of the 7th International Conference on The Semantic Web*, ISWC '08 (pp. 17–32). Berlin, Heidelberg: Springer-Verlag.

[Uren et al., 2006] Uren, V., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E., & Ciravegna, F. (2006). Semantic annotation for knowledge management:requirements and survey of the state of the art. *Web Semantics: Science, Services and Agents on World Wide Web.*, 4(1), 14–28.

[Vallet et al., 2005] Vallet, D., Fernández, M., & Castells, P. (2005). An ontology-based Information Retrieval Model. In *Extended Semantic Web Conference*, volume 3532 (pp. 455–470).

[Volz et al., 2003] Volz, R., Oberle, D., Staab, S., & Motik, B. (2003). Kaon server - a semantic web management system. In *Alternate Track Proceedings of the Twelfth International World Wide Web Conference, WWW2003, Budapest, Hungary, 20-24 May 2003*: ACM.

[Wilcock, 2009] Wilcock, G. (2009). *Introduction to Linguistic Annotation and Text Analytics*. Morgan & Claypool Publishers, 1st edition.

[Wu et al., 2007] Wu, J., Holt, R., & Hassan, A. E. (2007). Empirical evidence for SOC dynamics in software evolution. In *Proceedings of the International Conference on Software Maintenance* (pp. 244–254).: IEEE Computer Society.

[Xuan et al., 2006] Xuan, D. N., Bellatreche, L., & Pierra, G. (2006). A versioning management model for ontology-based data warehouses. In *International Conference on Data Warehousing and Knowledge Discovery(DaWaK)* (pp. 195–206).

[Zablith, 2008] Zablith, F. (2008). Dynamic ontology evolution. In *International Semantic Web Conference Doctoral Consortium*.

[Zablith et al., 2008] Zablith, F., Sabou, M., & Motta, E. (2008). Using background knowledge for ontology evolution. In *Proceedings of the ISWC International Workshop on Ontology Dynamics (IWOD)*.

[Zhang et al., 2010] Zhang, H., Li, Y.-F., & Tan, H. B. K. (2010). Measuring design complexity of semantic web ontologies. *J. Syst. Softw.*, 83(5), 803–814.

[Zhang et al., 2008] Zhang, L., Xia, S., Zhou, Y., & Xia, Z. (2008). User defined ontology change and its optimization. In *Control and Decision Conference, 2008. CCDC 2008. Chinese* (pp. 3586 –3590).

Appendix A

Software Help Management Case Study

A.1 Introduction

This case study covers a software help management domain. A software help system is part of a software product which focuses on delivery of help to users of a software product. Software products are released with associated help files that explain the overall purpose of the software and its components. The help files describe the product, its purpose, the components in the software, the tasks, the procedures and steps required to use the software and to troubleshoot a problem. The help files are prepared by professional software developers to assist users to efficiently use the software. The help files include description about concepts used in the software, locations of items and GUI elements, steps and procedures, shortcut keys and a lot other information useful to users of the product.

Software help files are content files which describe a software. The help files are created by domain experts who specialize in the domain of the software product. These files contain structured content in an XML format or semi-structured content in HTML format. Help files are released together with the associated software or become available on the web for on-line users. Searching, indexing, and browsing based on a table of contents which organizes the content using DocBook structures are used to efficiently access a specific content.

Software help files evolve dynamically. Whenever a software evolves due to changes, the associated help files evolve together. The evolution is in response to the changing behaviour of the software system. The evolution in the content affects the ontologies. The ontologies reflect the current semantics in the content and in the software. They further reflect existing structure of concepts in both the content and the software. Thus, when a software evolves, the help files evolve together. This causes the ontologies to evolve in turn propagating all the changes to annotations and dependent systems.

Our empirical study uses Symantec Enterprise Vault (EV) software help files. These help files are prepared by Symantec Enterprise Vault software developers. The help files further contain major concepts used in the software. The developers ranked concepts based on percentage of similarities among each other, and organized them using concept taxonomies.

For this study, we gain access to two versions of the EV software help files. The versions are EV version 7 and EV version 8. Version 7 contains 162 HTML files organized into 4 folders. The folders contain help files which represent four components of the software. Version 8 contains 839 files organized in 17 folders. The folders are used to categorize the help files based on the available software components.

Using the information provided on the help files and the associated concepts, we build ontologies which describe the overall structure and semantics of the software system. We developed four ontologies representing different aspects of the system. A high level view of the ontologies is given in Figure A.1.

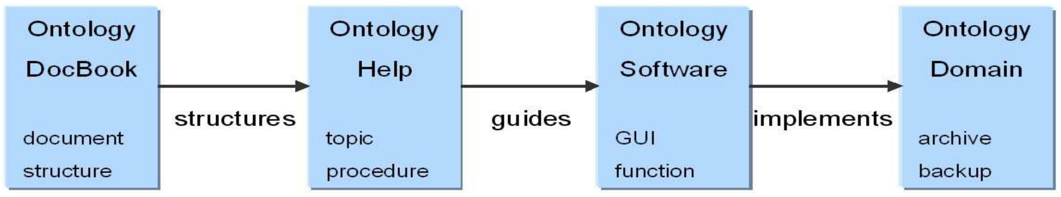


Figure A.1: A high level view of software help management ontology

The first ontology is the DocBook ontology which describes the overall structure of the help files [Pahl et al., 2010]. The DocBook ontology is constructed by extracting the structural entities from available DocBook files. A similar approach has been used to automatically extract a DocBook ontology from the DocBook structure [Şah & Wade, 2010] and XML files [Ceravolo et al., 2007]. The second ontology is the help ontology. The help ontology is designed to guide the software ontology by providing semantics about the help files and their content. The third ontology is a software ontology which is constructed considering a general software system specification. This ontology is used to describe the different behaviours and components of a standard software. It provides semantics about software related concepts. The fourth is a domain ontology which specifically focuses on the domain area of a software at hand. The domain ontology is also known as application ontology. In our case study the domain of the software is digital archiving which includes backup, searching, sharing, etc.

These ontologies are constructed using different information sources such as the different versions of the software help files, topic matrix, the document structure, and so on. A snapshot of the ontology is presented in Figure A.2.

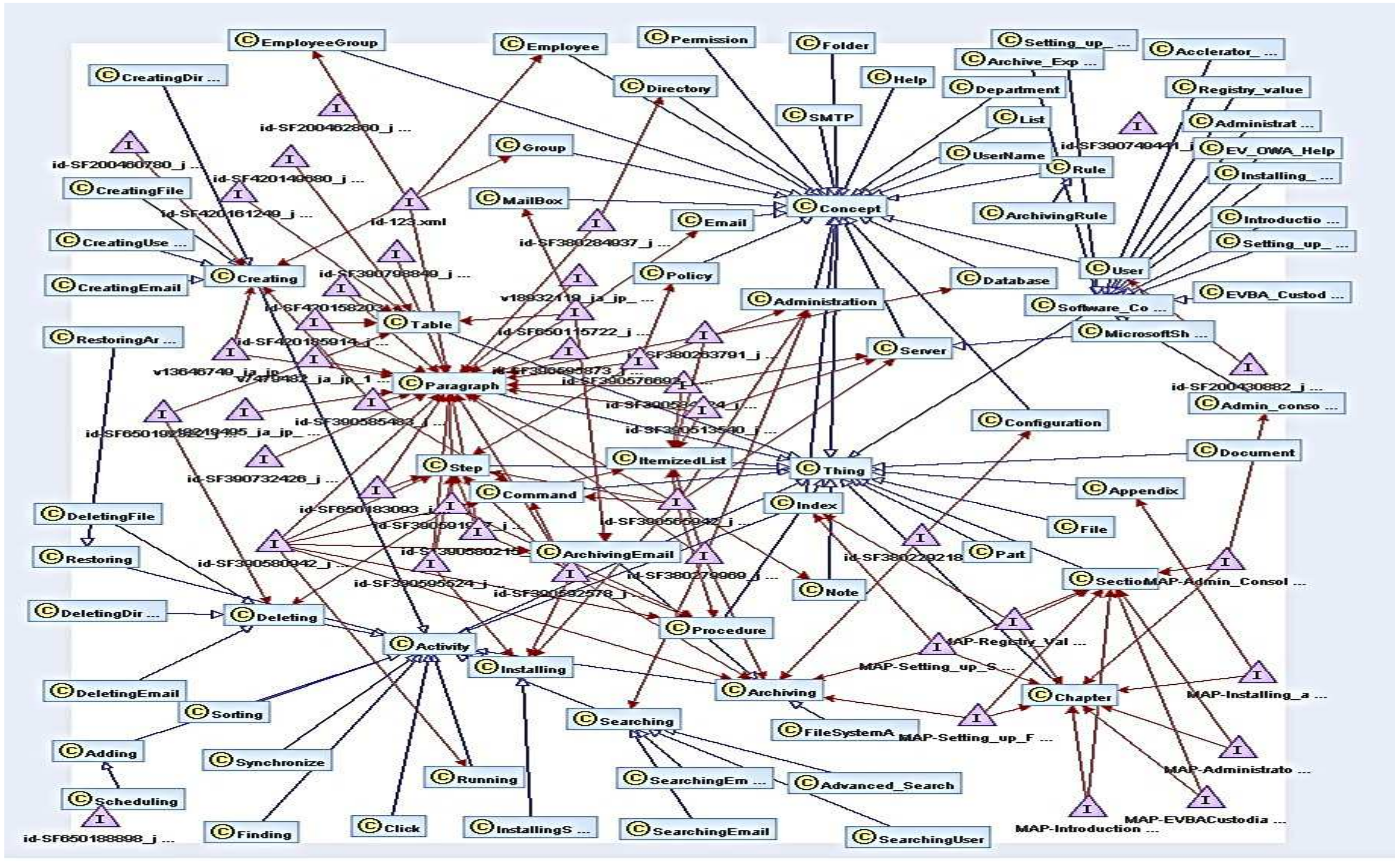


Figure A.2: Software help management ontology hierarchy

A.2 Rationale and Significance of the Case Study

This case study is selected because it has a wide coverage of topics that range from the application domain to software systems domain. There are different software tools in different disciplines covering different subjects within the disciplines. In these software products, the help files are associated to the knowledge and activities representing the disciplines. Help files are organized using document structures borrowing concepts from other domains. They further contain software concepts such as GUI elements, commands, hardware and software requirements, etc. This case study covers one or more of the above concepts.

Moreover, the concepts and instances are distributed throughout the help files and create a strong link between the instances in the content and the concepts in the ontologies. This makes it of great interest to investigate instance-driven change impacts because the changes made in the content of the help files will have an impact on the ontology and vice versa.

This case study is different from the other two case studies. First, it focuses on changes in both the content and the ontology. The changes in the content of the help files or the software cause a change in the ontologies. Second, we have different domain ontologies. These ontologies are interrelated and interdependent. Third, the individual help files are treated as individuals that are used to explain one or more concepts. These individuals are linked to different concepts from the ontologies. These features make this domain suitable to investigate the effects of changes in ontologies or in content documents.

The purpose of this case study is to investigate changes and the impacts of changes in the software help files in different versions of a software. This is part of the wider investigation of changes in different versions of an OCMS. In this case study, we specifically focus on investigating the following issues.

- Identification of changes from the versions which affect other layers of an OCMS.
- Investigation of effects of these change operation.
- Identification of dependencies that characterize the propagation of effects of changes to dependent entities and systems.

- Investigation of implementation options using different strategies and investigating of important criteria which can be used to to identify optimal implementation strategies.

A.3 Experiment

We used different perspectives for a better understanding of the problem and the behaviour of the evolution of an OCMS system. These perspectives are used to investigate changes at different layers, the strategies used to handle the changes and the different effects of the evolution process. The perspectives are discussed as follows.

A.3.1 Perspectives

The first perspective deals with the organization of the help file using DocBook structures. It organizes the help file into chapters, sections, paragraphs [Şah & Wade, 2010]. It provides steps, procedures, tips to perform a task. Tables, figures, and demonstrations are also included in this perspective. The DocBook ontology gives structure to and defines how elements in the help ontology are organized.

The second perspective is the help management systems perspective focusing on providing information on what a given concept is, how it is used. It explains how the software ontology makes use of the topics, procedures, etc. The help ontology guides the software ontology in a way that explains the system. What makes this perspective different from the others is, it focuses on help contents specific to the software at hand.

The third perspective is the software systems perspective. This perspective views the help files as part of a software functionality which provides better facility for using an application. It provides explanations, steps and procedures on how to use a specific part of the application, where to find a GUI button, how to activate a given process, etc. This perspective focuses on generic software features that are available in different software systems.

The fourth perspective also focuses on the organization of the components of the software in a given domain. For a software, we explain the different subcomponents of the software, what the individual components do and how they are related to each other. For example, in the case of EV customer help, we have different subcomponents that deal with emails, files, folders etc. The perspective allows us to implement the ontology domain which is specific to the components in each application. These components are organized based on the concept structures in the domain or based on the activities supported in the software.

A.3.2 Changes

We studied the changes from one version to another version and trace those changes to find the frequency, the importance and the impacts of the changes. We identified several changes but focused on 15 selected scenarios that cover all the four ontologies. These scenarios represent the different evolutions in the OCMS. The selection of each scenario is based on the frequency of the change, their cascaded impacts, the operations involved and the number of ontologies affected. The analysis of the change scenarios exposes different kinds of changes.

A.3.2.1 Changes among Versions

We compare the two versions of the help files (version 7 and version 8). Following the analysis result, we categorize the changes into three different categories.

Changes in the Content. A change in the content refers to the change in the actual contents of the help files. This includes the introduction of new help files to support the new components and functionalities of the new version of the EV software. Some of these changes are the addition of the *Accelerator_Client_Help*, *Administrator_Guide* and so on. All these components have their own associated help files that are introduced in the new version. Old concepts which are not included in the new version are also removed.

Changes in the Structure of the Docbooks. These categories of changes are identified in the new version. This is due to the fact that the whole help file structure is changed from HTML based help files to XML based help files. Because of this change, all the sections, subsections, paragraphs, tables, lists, etc., are introduced in the DocBook by incorporating new tags that were not available in the previous versions.

Changes in the Formats of the Files. These categories of changes are identified when the formats of the help files evolve and include pictures, graphs, charts, and most importantly video and audio help files that illustrate steps and procedures on how to perform tasks. To illustrate the above changes, we use the following two scenarios discussed in [Abgaz et al., 2010]

Scenario 1. The new version of the software resulted in a change of component *X* which contains other two sub components *Y* and *Z*. The component *X* and its subcomponent *Y* are removed but the subcomponent *Z* is moved up. Here, all the previous instances (help files) of *X* and *Y* are preserved as instances of *Z*. *X*, *Y* and *Z* stand for different components that change following a similar pattern. The desired output is an updated ontology which reflects the requested change. The change operations are:

- Move up (*Z*)
  - Add instance of (*instance of X*, *Z*) . . .
  - Add instance of (*instance of Y*, *Z*) . . .
- Delete concept (*Y*)
- Delete concept (*X*)

Scenario 2. The software engineers introduced a new software component *NC*. The new component has new associated help files. The desired output is a software application ontology that has a description of the new component and its properties. The change operations are:

- Add Concept (*NC*)
- Add sub concept (*NC*, *softwareApplication*)
- Add instance (*Help_file*). . .
- Add instance of (*Help_file*, *NC*)

A.3.2.2 Changes within a Single Version

Software life cycle involves different development stages. At each stage of the life cycle there are changes. At the construction stage of the software, different new components will be included. The help files and documents of the respective components are also produced in parallel. As the software components passes through different stages, the documentation and the help files also update synchronously. Furthermore once the software product at a certain stable version is released, it is not an exception to find subversions and patches following the release of that version. With such subversions and patches, the help files and associated ontologies that are used to support the software system need to be updated.

Scenario 3. The enterprise vault software engineers split the *Backup and restore* menu item into two separate menu items: back-up, and restore menu items for simplicity of use within the existing version. The change is represented as follows.

- Split concept (*Backup_and_Restore*, *Backup*, *Restore*)
    - Add concept (*Backup*) . . .
    - Add concept (*Restore*) . . .
- Delete (*Backup and Restore*)

From the software help management case study and following the above procedure, we identified 15 frequent change scenarios to evolve the system. The scenarios are selected based on their frequency and the number of ontologies they affect in the system. They also represent the deletion and addition operations and mix them evenly. Their effect is also taken into consideration to identify the changes. The overall changes are presented in Table A.1

Table A.1: Change scenarios

No	**Change Scenario**	**Change Operations**	**Affected Ontology**	**Impact Type**	**Frequency of change**
1	The enterprise vault software engineers introduced a new software component. The new component has associated help files. **Desired output :** the software application ontology needs to have a description about the new component and its properties.	- Add Concept (*new component*) - Add sub concept (*new component, software Application*) - Add instance (*Help_file*) - Add instance of (*help_file, new component*)	Application Ontology	Structural and Semantic	Very Frequent
2	The new version of the Enterprise Vault software application has made a change on a component which contains two other sub components. The component and one of its subcomponents are removed but the other subcomponent is upgraded as a full component. Here we do need to link all the previous instances associated with the removed concepts to the upgraded concept. **Desired output :** an updated ontology which reflects the requested change.	Move up (*subcomponent2*) - Add instance of (*instance of component, subcomponent2*) - Add instance of (*instance of subcomponent1, subcomponent2*) - Delete concept (*Component*) - Delete concept (*subcomponent1*)	Application Ontology	Structural and Semantic	Very Frequent
3	Replacing function by software feature in the application ontology. **Desired output :** the function is changed to software feature in the ontology.	- Add concept (*software feature*) - Add subclass of (*software feature, super class of function*) - Add subclass of (*subclass of function, software feature*) - Delete concept (*function*)	Application Ontology	Structural and Semantic	Frequent

4	A new reference mechanism (eg file_id, file_uri) is added to identify help files in the help management and cross references. **Desired output :** the new properties (file_id and file_URI) are added into the DocBook ontology.	- Add data property (*file id*) - Add data property (*file uri*) - Add range (*file_id, Integer*) - Add range (*file_uri, String*)	Application and DocBook Ontology	Structural and Semantic	Frequent
5	The software engineers added additional help formats (video and audio) to enhance the functionality. **Desired output:** the ontology help infrastructure needs to introduce a new concept format.	- Add concept (*format*) - Add sub class of (*format, some super class(y)*) - Add subclass of (*sub class of (z) the super class(y), format*) - Add instance of (*instance of format, format*)	Help and DocBook Ontology	Structural and Semantic	Medium
6	In the previous version of the software *paragraph* was treated as a direct sub concept of *chapter* concept. However in the new version of the help file it is treated under section. **Desired output :** the help infrastructure ontology removes paragraph from chapter and redirect it to section.	- Add subclass of (*paragraph, section*) - Delete subclass of (*paragraph, chapter*)	Help Ontology	Structural and Semantic	Medium
7	In the new version, the concept *query* and *topic* are merged together and are called *subjects*. **Desired output :** the concept *query* and *topic* and all their properties and instances merged into *subject* in the software system ontology	Merge concept (*topic query, concept query, subject*) - Add concept (*subject*) - Add subclass of (*topic_query subclass, subject*) - Add subclass of (*concept_query subclass, subject*) - Add instance of (*topic_query instance, subject*) - Add subclass of (*concept_query instance, subject*) - Delete concept (*topic_query*) - Delete concept (*concept_query*)	Help Ontology	Structural and Semantic	Medium

8	The new version of the enterprise vault included snapshots as a picture in the help file. A single picture can be used one or more times to illustrate steps and procedures. Pictures have descriptions like picture number and name. **Desired output :** the concept picture and its properties are included in the software system ontology	- Add concept (*picture*) - Add subclass of (*some super class*) - Add property (*picture_number*) - Add property (*picture_name*) - Add domain (*picture_number, picture*) - Add domain (*picture_name, picture*) - Add range (*picture_name, String*) - Add range (*picture_number, Integer*)	Help Ontology	Structural and Semantic	Less Frequent
9	The enterprise vault software engineers have changed some of previously known functionalities provided by buttons into functionalities provided by menus. Furthermore they removed the buttons that were providing those functionalities. **Desired output :** all the buttons need to be removed from the ontology.	- Delete concept (*buttons*)	Software Ontology	Structural and Semantic	Frequent
10	Continuing from the above (6), the contents (instances associated with the removed buttons) are redirected to the menu items. **Desired output :** all the instances that are linked to the buttons should be linked to the respective menu.	- Add instance of (*instances of button", "Menu*)	Software Ontology	Structural and Semantic	Frequent
11	The enterprise vault software engineers removed an old top level menu item. **Desired output :** the menu item is removed from the ontology, and its sub classes linked to the appropriate super class.	- Delete concept (*top level menu item*)	Software Ontology	Structural and Semantic	Medium

12	The enterprise vault software engineers split the *backup and restore* menu item into *back-up*, and *restore* menu items. **Desired output :** the two concepts treated separately as siblings.	Split concept (*backup and restore, backup, restore*) - Add concept (*backup*) - Add concept (*restore*) - Delete (*backup and restore*)	Software and Application Ontology	Structural and Semantic	Frequent
13	Removing a certain GUI feature (toolbar). **Desired output :** the GUI feature and its instances are removed from the software system ontology.	- Delete concept (*toolbar*) - Delete instances (*instance of toolbar*)	Software Ontology	Structural and Semantic	Frequent
14	The new versions of the software removed the command line features of the software and its associated contents. **Desired output :** the concept command line is removed from the software ontology with all its instances.	- Delete concept (*command_line*) - Delete concept (*sub class of command_line*) - Delete individual (*individual of command_line*)	Software Ontology	Structural and Semantic	Less Frequent
15	When a new version of the software is released, the structure of the DocBook has been changed into a different structure. The new structure pulled up some of the elements and merged the others. Some of the elements are also split into different elements. **Desired output :** the changes are implemented according to the requested changes.	- Pull concept up (*some concept*) - Merge concept (*concept1, concept2, concept3*)	DocBook Ontology	Structural and Semantic	Less Frequent

We implement changes based on the scenarios and evolve the OCMS accordingly. The summary of our observation on the changes, dependencies and impacts is presented in the following section.

A.4 Observation

Using the above change categories we identify individual changes and further analyse the frequency, the complexity and the error prone nature of the changes. We investigate the types of impacts the changes have on the overall content-based system.

We find changes that are occurring frequently whenever there is a new subversion of the software component. The phenomenon that requires an intensive research focus is the complexity of the changes, and the propagation of their effects. Many of the changes in such systems have effects that propagate to other components due to the interdependence between the components of the systems.

In this case study we observe that one help file can be used to illustrate tasks of one or more components of the system. One component of the system is linked to two or more other components. If there is a change in one of such components, it will suddenly become complex and beyond the comprehension of the content manger or the ontology engineer to find all the dependent components and to see the changes that propagate to other components.

We identify such dependencies between elements of the ontology, the content and the annotations that define the relationship between the content and the ontologies. These dependencies explain the impacts of a change on other components and the nature of propagation of the impacts.

In addition to this, the interdependence that exists between the content and the ontology is very high (as the ontologies are built to reflect the interaction and the structure of the software system at hand). This means, a change in one or more of the software component hierarchies will affect the ontology taxonomy and vice versa.

Dealing with such kinds of changes in the ontology manually is beyond the capacity of both the ontology engineer and the content manager. In software systems that have large number of components, developers require an intensive tool support for handling the changes and analysing and determining the impacts of the changes. From the software point of view, to support such systems, there are different tools for controlling different versions such as CVS and others. However, from OCMS's view point, these problems are not yet solved. This case study reveals the following major problems and explains the real world challenges associated with OCMS.

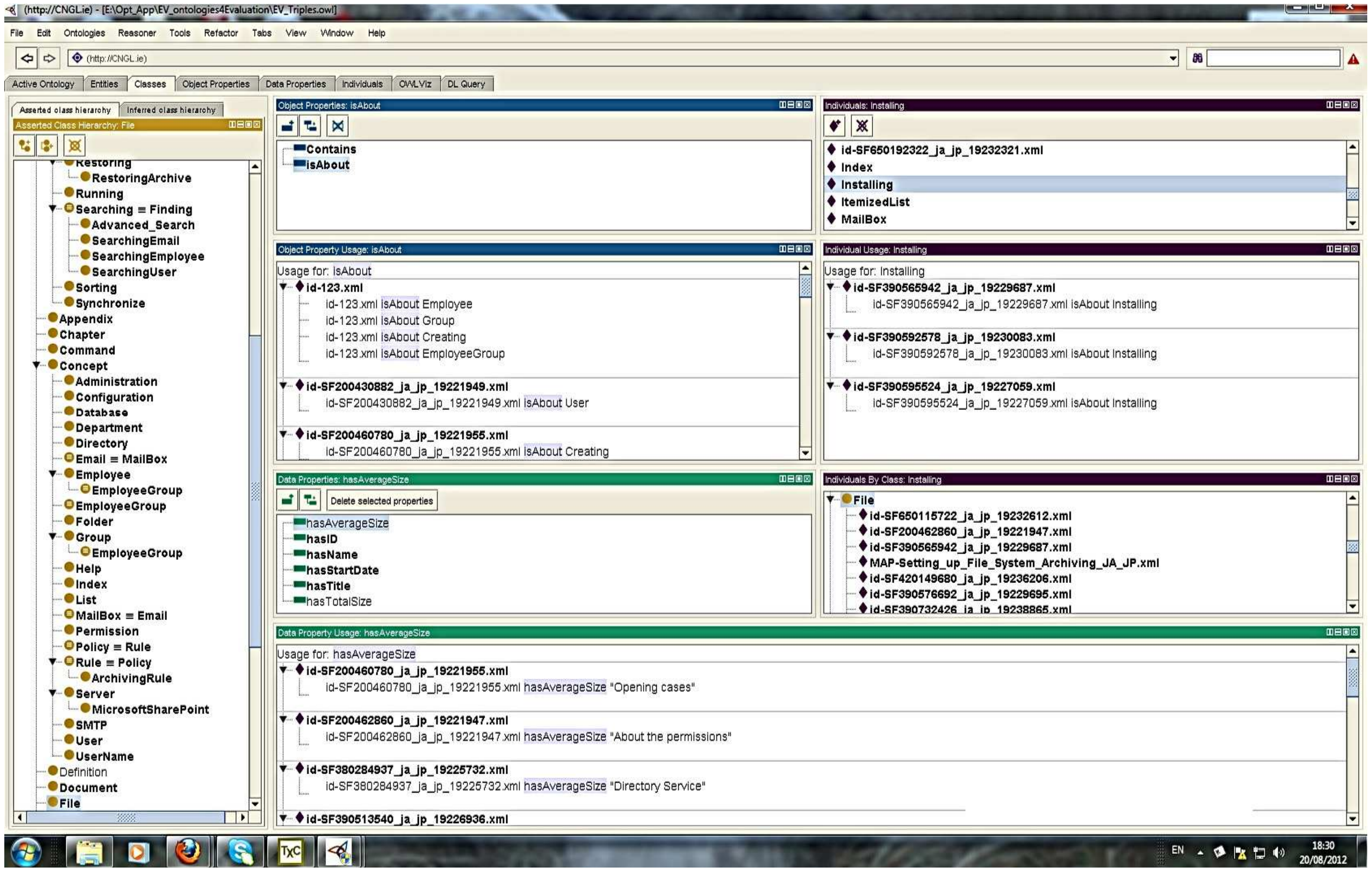
1. Characterizing changes. This case study explains how content-based systems evolve by analysing different versions of software help management system. The changes are not restricted to software help management files but include other related applications. We identify basic changes that add new concepts and remove existing concepts. All other changes can be explained by a series of additions and deletions composed in a certain way. Many of the changes involve either addition or removal of new instances, concepts, and properties of concepts or their descriptions. We have also observed that changes include restructuring existing entities and splitting or merging of existing concepts, etc., which can be considered as composite changes.

2. Representation of changes. Using the ontology and the content, the case study served as a means to represent the actual changes using change operations defined in the OCMS. The changes can be represented at different levels of granularity which includes atomic level and composite level changes. It is evident that a given change can be represented using different compositions of the atomic change operations. It also reveals the different strategies of implementing change operations. From the concrete scenarios, we identify strategies that cascade the changes to all dependent entities, only to the instances or apply the change only on the target entity and link the dependents to the root or the parent entities. Such scenarios serve as a justification for the implementation of a change operation using different strategies based on the requirements of the user. To meet the requirements of users, OCMS should provide such flexibility to the users.

3. Identification of dependencies. The case study clarifies the importance of deeply understanding the dependencies between entities in the OCMS system. This enables us to efficiently implement the change operations and select a strategy that meets the requirement of the user. We identify direct dependencies, indirect dependencies and partial and total dependencies between entities. The case study provides us real world scenarios that distinguish dependencies between different types of entities.

4. Identification of impacts and nature of impacts. This case study further discovers the different impacts of change operations. Some of the change operations have structural impacts which affect the taxonomy of the OCMS and others affect the semantics of the concepts and instances. We identify different structural impacts and semantic impacts using the case study. Explaining the impacts of the change operations and identifying the preconditions and the affected entities of the change operations provide crucial information for selecting the optimal implementation from different options. Finally, the case study revealed that which entities are important in the OCMS and need to be preserved and which ones are less important and can be sacrificed. This includes the $\mathcal{AB}$ox versus $\mathcal{TB}$ox statements, additions versus deletions and unsatisfiabilities and inconsistencies.

A.5 A Snapshot of Software Help Management Ontology



(http://CNGL.ie) - [E:\Opt_App\EV_ontologies4Evaluation\EV_Triples.owl]

File Edit Ontologies Reasoner Tools Refactor Tabs View Window Help

(http://CNGL.ie)

Active Ontology | Entities | Classes | Object Properties | Data Properties | Individuals | OWLViz | DL Query

Asserted class hierarchy | Inferred class hierarchy

Asserted Class Hierarchy: File

```
Restoring
    RestoringArchive
  Running
  Searching ≡ Finding
    Advanced_Search
    SearchingEmail
    SearchingEmployee
    SearchingUser
  Sorting
  Synchronize
Appendix
Chapter
Command
Concept
  Administration
  Configuration
  Database
  Department
  Directory
  Email ≡ MailBox
  Employee
    EmployeeGroup
  EmployeeGroup
  Folder
  Group
    EmployeeGroup
  Help
  Index
  List
  MailBox ≡ Email
  Permission
  Policy ≡ Rule
  Rule ≡ Policy
    ArchivingRule
  Server
    MicrosoftSharePoint
  SMTP
  User
  UserName
Definition
Document
File
```

Object Properties: isAbout

```
Contains
isAbout
```

Object Property Usage: isAbout

Usage for: isAbout

```
id-123.xml
    id-123.xml isAbout Employee
    id-123.xml isAbout Group
    id-123.xml isAbout Creating
    id-123.xml isAbout EmployeeGroup
id-SF200430882_ja_jp_19221949.xml
    id-SF200430882_ja_jp_19221949.xml isAbout User
id-SF200460780_ja_jp_19221955.xml
    id-SF200460780_ja_jp_19221955.xml isAbout Creating
```

Individuals: Installing

```
id-SF650192322_ja_jp_19232321.xml
Index
Installing
ItemizedList
MailBox
```

Individual Usage: Installing

Usage for: Installing

```
id-SF390565942_ja_jp_19229687.xml
    id-SF390565942_ja_jp_19229687.xml isAbout Installing
id-SF390592578_ja_jp_19230083.xml
    id-SF390592578_ja_jp_19230083.xml isAbout Installing
id-SF390595524_ja_jp_19227059.xml
    id-SF390595524_ja_jp_19227059.xml isAbout Installing
```

Data Properties: hasAverageSize

Delete selected properties

```
hasAverageSize
hasID
hasName
hasStartDate
hasTitle
hasTotalSize
```

Individuals By Class: Installing

```
File
  id-SF650115722_ja_jp_19232612.xml
  id-SF200462860_ja_jp_19221947.xml
  id-SF390565942_ja_jp_19229687.xml
  MAP-Setting_up_File_System_Archiving_JA_JP.xml
  id-SF420149680_ja_jp_19236206.xml
  id-SF390576692_ja_jp_19229695.xml
  id-SF390732426_ja_jp_19238865.xml
```

Data Property Usage: hasAverageSize

Usage for: hasAverageSize

```
id-SF200460780_ja_jp_19221955.xml
    id-SF200460780_ja_jp_19221955.xml hasAverageSize "Opening cases"
id-SF200462860_ja_jp_19221947.xml
    id-SF200462860_ja_jp_19221947.xml hasAverageSize "About the permissions"
id-SF380284937_ja_jp_19225732.xml
    id-SF380284937_ja_jp_19225732.xml hasAverageSize "Directory Service"
id-SF390513540_ja_jp_19226936.xml
```

EN 18:30 20/08/2012

Figure A.3: Software help management systems OCMS

Appendix B

Database Systems Course Case Study

B.1 Introduction

This case study focuses on an e-learning system that deals with a database systems course. The database systems course covers theoretical, practical, technological aspects of database design, implementation and management. Database systems courses introduce such knowledge to the students by providing books, course notes, tutorials, manuals, demonstrations, exercises, questions and sample exams and answers. It further covers new research and future directions in the field of database systems and related domains.

The content in this domain is created by different stakeholders. Instructors of the course may compile lecture materials, authors write books, software development companies prepare software and produce books, help files and documentations for the software, etc.

The evolution in database systems domain is dynamic. There are different conceptual models introduced since the inception of the concept, design specifications are proposed, languages are created. Software systems that support both the design and implementation are produced and are evolving dynamically to support the current requirements. A typical evolution related to database models starts from hierarchical data model to network model then to relational model. Now the relational data model evolves to support object oriented specifications. The domain continues to evolve to support current requirements and technological advancements in the field of computer science.

This case study covers a domain which focuses more on conceptualization. This means, the database course ontology defines concepts that are used in database systems. This domain enables us to cover OCMS that focuses more on $\mathcal{TB}$ox statements than $\mathcal{AB}$ox statements. In this OCMS, the definition of the concepts in the domain are much important than the individuals associated with those concepts.

B.2 Case Study Setup

Similar to the other case studies, we explore different perspectives to understand how the domain changes and what changes are observed frequently. We identify different perspectives but categorize them into three.

B.2.1 Perspectives

The Publishing Perspective. The publishing perspective focuses on preparation and publishing of content in different media such as books, video tutorials, etc. We study the publications related to database systems course. The database course is among the courses that exhibit frequent changes, updates and modifications. The database ontology was derived from the taxonomy arising from the table of content and the indexes of the text books. In the past 20 years, there are lots of changes observed [Javed et al., 2009]. Addition of new concepts, technologies, techniques, applications and languages and removal of obsolete concepts and modification of existing ones are observed frequently. These frequent changes cause the database systems OCMS to evolve frequently.

The situation gets worse when the course content is subject to modification by different stakeholders in different places. A database system book writer views the changes as changes in taxonomy represented in the table of content or in the index at the end of his book. A concept treated under one section can be moved to another section or two concepts merged together and create a new concept. The following examples represent changes in the taxonomy of table of content [Javed et al., 2009].

Atomic changes

- create concept *SQL* using Create Concept (*SQL*)
- make *SQL* subconcept of *Database* using Subconcept(*SQL, Database*)

Composite changes

- Merge Concept (*DDL, DML, Database*):
  - Integrate Concept Context by creating concept *Database* using Create Concept (*Database*)
  - Integrate Property Context by creating property *isBasedOn* using Create Property (*isBasedOn*)
  - Integrate Domain/Range Context by adding a domain to *isBasedOn* using Add Domain*(isBasedOn,Database)*
  - Adding range *Relational Algebra* to *isBasedOn* using Add Range *(isBasedOn, RelationalAlgebra)*
  - Remove Concept Context by deleting concept *DML* using Delete Concept*(DML)*
  - Deleting concept *DDL* using Delete Concept*(DDL)*

The Technology Perspective. Technological advancement in the area causes a change in the overall structure and semantics of the content and the ontologies. Now days, the change in the technology is dynamic that a continuous update is required to ensure the up-to-datedness of such content-based systems. The emergence of object-oriented databases, web-based databases and a lot other technologies and software every time cause systems to update themselves synchronous with the changes.

Example:

- Add Concept*(Web-BasedDatabase)*
- Add subclassOf*(web-BasedDatabases, Database)*

The Teaching Perspective. In the teaching and research perspectives, there are huge number of course outlines produced which differ in their depth and coverage. These outlines are also subject to modification each semester. In addition to this, there are a number of new research papers contributed in the area, reports about new tools, techniques and technologies, and a number of case studies, experiments, etc., that adds new knowledge to the domain. Professors prepare examples, mock questions and examinations that serve as instances of concepts. These materials change frequently and contribute for the evolution of the system. For example, the teacher wants to merge relational algebra and relational calculus and give it a name relational operation.

- Merge concept *(RelationalAlgebra, RelationalCalculus, RelationalOperations)*

To implement an efficient content-based system, which can deal with such dynamic content, it requires a careful understanding of the ongoing changes, a detailed analysis and a sound solution that reflects and propagate changes to dependent elements of the system. To achieve such requirements, we need to deal with the following challenges. The first challenge is the frequency of the changes. The changes in such systems are continuous that we need to provide a continuous means of tracking, understanding and implementing the changes. For example a continuous release of new research papers that contains new topics in the area needs to be incorporated as soon as the contributions are accepted by the scientific community and domain experts in the area.

The second challenge is the volume of change. For relatively large content-based systems, the amount of change the systems deal with is numerous. Especially for systems that are used in a multi user environments where different stakeholders change the content and others access the same content, the amount of change that needs to be dealt will become very large and beyond the comprehension of the users. The third challenge is the complexity of the change. It requires a detailed research to understand the effects of the changes on other entities. There are changes that can be implemented easily and there are changes that are complex and require an expert involvement to implement them.

Such kinds of changes are not only complex by themselves but also have complex cascaded effects on the dependent systems and contexts. The complexity of the changes leads to errors. As the changes and the cascaded effects of those changes become complex, the chance of introducing error into both the structure and the semantics of the system will increase. That leaves the systems inaccurate and unreliable.

- merge two concepts in one concept
  - create a new concept.
  - add the new concept as a sibling concept
  - add subclasses of the concepts to the new concept
  - add instances of the concepts to the new concept
  - add properties of the concepts to the new concept
  - delete the old concepts
- split a concept into many concepts
  - add the new concepts as sibling
  - add the subclasses of the split concept to the sibling
  - add the instances of the split concept to the sibling
  - repeat the above steps for all the sibling concepts

Example:

- Merge concept*(Relational_Algebra, Relational_Calculus, Relational_Operations)*

When we look at the above operation, it seems a single and a simple operation. However, it contains many atomic change operations in it. The merging of relational algebra and relational calculus involves the merging of all their subclasses too. While doing that we have to look at axioms and constraints that should be kept valid before and after merging.

B.3 Experiment

We conducted empirical investigation to understand what changes are observed and which entities are changing frequently. The experiment tries to identify the changes from each perspective. The partial representation of the ontology hierarchy is presented in Figure B.1

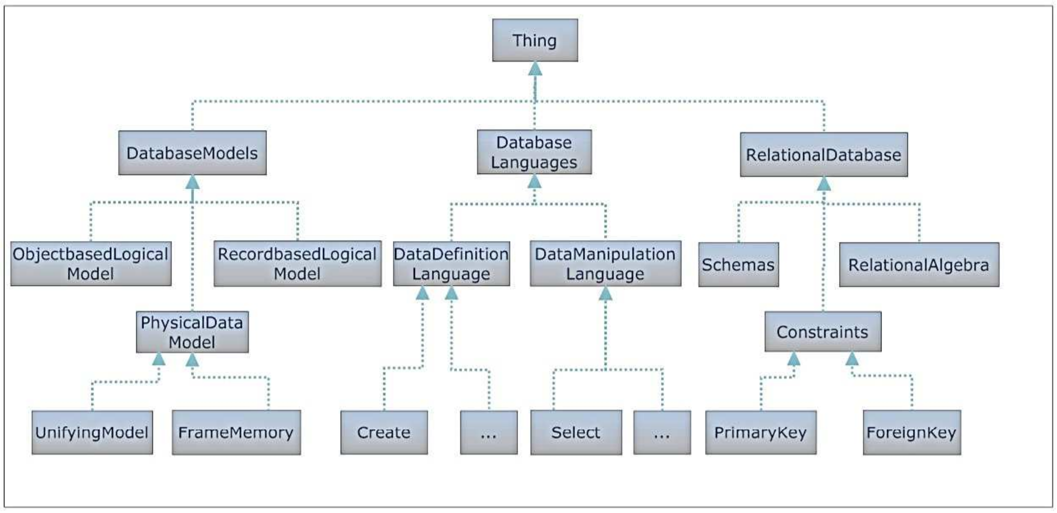


Figure B.1: A high level view of database ontology

B.4 Observation

In publishing perspective, we identified changes from different versions of published text-books over the past 20 years. The changes can be generalized as changes that include emerging concepts, changes that remove obsolete concept, and changes that merge or split existing concepts.

In publishing, the changes are more related to the taxonomy and relationships between entities. The publishing contributes much content to the content layer.

From technology perspective, we identified changes that are related to introduction of new technology, what it does and how it does, where and when it is applicable. We further identify emerging technologies that are used to implement concepts discussed in the domain. Further obsolete technology solutions are removed from contemporary systems and are replaced by the new ones.

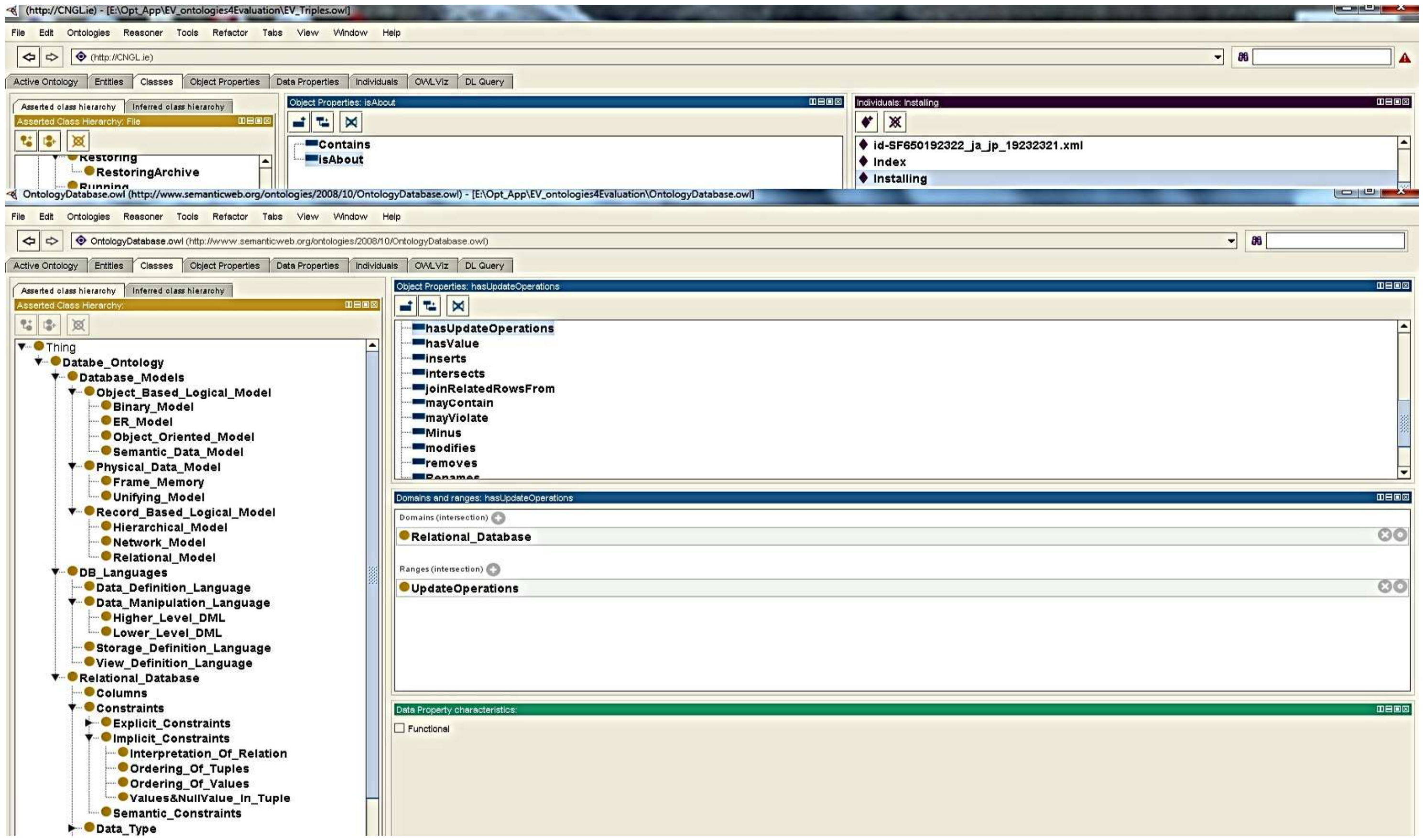
The changes in this perspective add new concepts and terminologies to the ontology.

This facilitates the annotation of the content with new terminologies.

The teaching perspective is the one that evolves frequently by covering new research results, new innovations and modifications of existing knowledge to fit for different perspectives depending on the aim of the course. In this perspective, we have changes whenever the semester changes or whenever the students change. The aim of the teaching perspective is to make students familiar to concepts and enable them to successfully work with the existing tools. This requires providing definitions, about the concepts, relationship between concepts and the properties of the concepts.

The teaching perspective incorporates changes in both the ontologies and the contents.

B.5 Sample Database systems course OCMS


(http://CNGL.ie) - [E:\Opt_App\EV_ontologies4Evaluation\EV_Triples.owl]
File Edit Ontologies Reasoner Tools Refactor Tabs View Window Help
(http://CNGL.ie)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query
Asserted class hierarchy Inferred class hierarchy
Asserted Class Hierarchy: File
Object Properties: isAbout
Individuals: Installing
OntologyDatabase.owl (http://www.semanticweb.org/ontologies/2008/10/OntologyDatabase.owl) - [E:\Opt_App\EV_ontologies4Evaluation\OntologyDatabase.owl]
OntologyDatabase.owl (http://www.semanticweb.org/ontologies/2008/10/OntologyDatabase.owl)
Asserted Class Hierarchy:
Object Properties: hasUpdateOperations
Domains and ranges: hasUpdateOperations
Domains (intersection)
Ranges (intersection)
Data Property characteristics:
Functional

```
Restoring
  RestoringArchive
Running
```

```
Contains
isAbout
```

```
id-SF650192322_ja_jp_19232321.xml
Index
Installing
```

```
Thing
  Databe_Ontology
    Database_Models
      Object_Based_Logical_Model
        Binary_Model
        ER_Model
        Object_Oriented_Model
        Semantic_Data_Model
      Physical_Data_Model
        Frame_Memory
        Unifying_Model
      Record_Based_Logical_Model
        Hierarchical_Model
        Network_Model
        Relational_Model
    DB_Languages
      Data_Definition_Language
      Data_Manipulation_Language
        Higher_Level_DML
        Lower_Level_DML
      Storage_Definition_Language
      View_Definition_Language
    Relational_Database
      Columns
      Constraints
        Explicit_Constraints
        Implicit_Constraints
          Interpretation_Of_Relation
          Ordering_Of_Tuples
          Ordering_Of_Values
          Values&NullValue_In_Tuple
        Semantic_Constraints
      Data_Type
```

```
hasUpdateOperations
hasValue
inserts
intersects
joinRelatedRowsFrom
mayContain
mayViolate
Minus
modifies
removes
Renames
```

Domains (intersection): Relational_Database

Ranges (intersection): UpdateOperations

Figure B.2: Database systems course OCMS

Appendix C

University Administration Case Study

C.1 Introduction

In this case study, we investigate a university administration domain. A university administration focuses on administration of large number of students, staff, departments, courses, and examinations. It manages campuses, buildings, class rooms to efficiently consume resources in a university. The case study incorporates different stakeholders of a university system such as students, research institutes, funding institutes, etc. The university system covers concepts, relationships and constraints on relationships and individuals. Unlike the previous case studies, this case study focuses more on instances and instance annotations. The annotations are used to semantically enrich related content with ontologies that represent the domain knowledge.

In this case study, the ontologies represent the organization structure, the course curriculum and the administration procedures. These ontologies are relatively stable and do not evolve frequently unless there is a structural change or a revision on the curriculum. However, the instances evolve dynamically. Since the registration of a new student to the graduation, the instance passes through different changes. The changes include additions of annotation data about the student, the department, the courses, the activities, etc.

This domain is selected for the following reasons. First, this domain represents a real world organization that evolves dynamically. The dynamic change in the domain allows us to investigate the changes. Second, this domain represents OCMS with large number of data instances. This corresponds to OWL QL profile which is optimized for domains with large number of instances. Unlike the other domains, the evolution in this domain primarily focuses on instances. Third, the evolution of the terminology affects large number of instances that are related to each other via relationships. This further enables us to study the impacts of the changes on the instances.

C.2 Case Study Setup

We modelled the university ontology using Dublin City University (DCU) as a case study. This case study allows us to explore $\mathcal{AB}$ox changes and their effects. Some of the features that make the university ontology suitable for the case study are discussed as follows. The university administration domain focuses on people, physical resources and events which are represented as assertion statements, thus represent more of the assertions than the terminologies. Many of the changes related to this domain focus on the instances. For example, a new faculty joins a department, a new student get registered and another gets graduated and all the related information of the student become inactive. In this domain, the ontologies are relatively stable, but the content evolves frequently causing the instances to change from time to time.

The ontology is created based on the perspectives of managing the university system. This enables us to restrict the ontology to focus on the major areas that are important for administering the university system and the proper execution of the day-to-day activities of a university.

Accordingly, we studied the different major entities and their relationships in a university system. The concepts give semantic definition for the actual entities involved in the administration process. For example, a concept researcher is defined in relation to person and someone who is conducting research. The concept researcher gives semantic definition to all individuals who are conducting research in the university. To take another example we have concepts such as department and course. These concepts provide semantic information about individual departments such as school of computing and school of engineering.

C.3 Experiment

For the purpose of the experiment we built a university ontology. The university ontology covers basic concepts such as faculty, department, course, staff, students and research groups. Each of these concepts is specialized into different specific concepts. The ontology further covers relationship between the concepts, domains and ranges of the relationships and identifies instance properties that apply for specific instances. The taxonomy of the ontology is presented in Figure C.1.

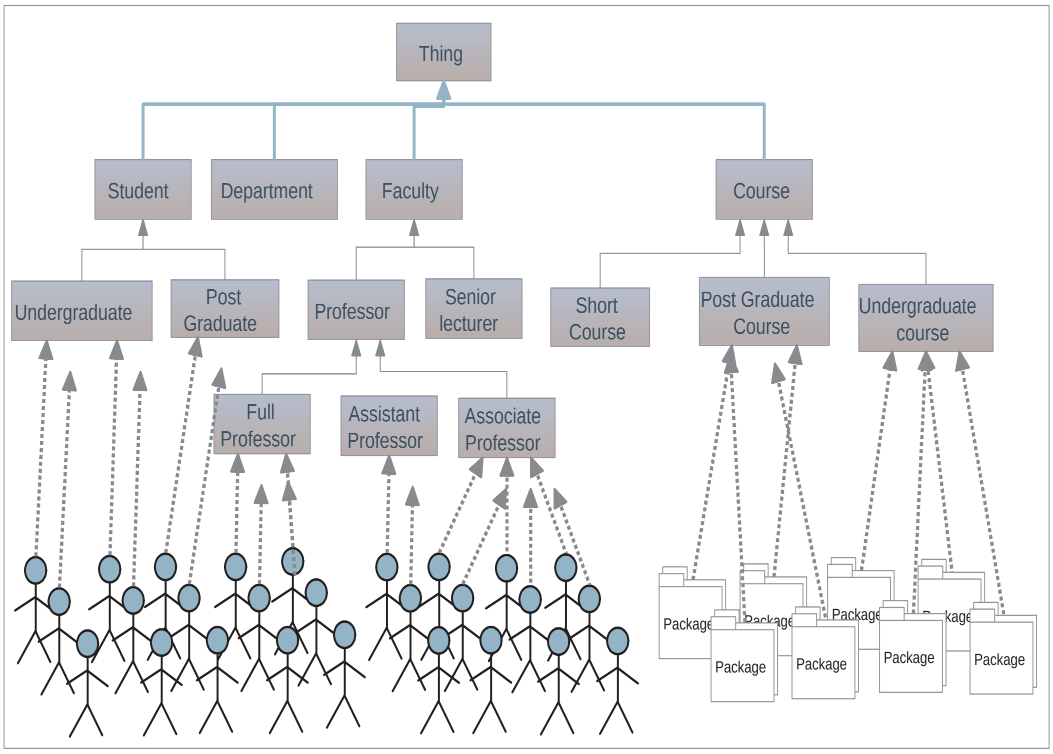


Figure C.1: University ontology hierarchy

Once the ontology is built, it passes through different evolutionary changes. The changes incorporate addition and deletions. These changes are discussed as follows.

C.3.1 Changes in University Administration

The changes in university administration come from changes in the conceptual definition or changes in the characteristics of the instances. Most of the frequent changes in this domain are related to instance changes. These instance changes include addition of new students, staff, department, etc., and deletion of obsolete instances. However, the conceptual definition of these instances changes seldom. We identified three different levels of changes in the university administration domain. These changes are presented here.

Level 1 is constructed to fit for lower level operation such as creating new concepts, deleting old concepts, etc., which are single and atomic tasks that change a single entity of the ontology. The atomic changes are presented using a natural language.

1. Create Instance
2. Make the instance an instance of a concept
3. Remove instance from the a concept
4. Create object property Assertion
5. Remove instance object property assertion
6. Create data Property assertion
7. Remove instance data property assertion
8. Add value of property from an instance
9. Remove value of property from instance
10. Change value of property from instance
11. Set the maximum cardinality of a property
12. Set the minimum cardinality of a property

Level 2 is constructed to fit for middle level operations like creating new instance, which may include different calls to the first level operations For example, creating a new instance involves, creating the instance as a new node and making it as instance of a concept. We present some of the change operations observed at this level.

1. Creating instance of *University* called *DCU*
    - Create instance *DCU*
    - Create instance of *(DCU, University)*

In principle, it is possible to create infinite number of level two change operations by combining the atomic change operations. But, in practice we use only those change operations that are used frequently to make changes.

Level 3 is constructed to fit for higher level operations like modifying the structure of the university administration, opening a new department or closing an old faculty. This level is constructed based on different perspectives we identified in the construction stage of the ontology. This level makes use of one or more operation from level 2. For deleting a certain super concept, we may need to delete the concept, remove the dependencies from the ontology, check the consistency, and amend the inconsistencies if they are introduced in the system. Sample lists of changes in this level are listed below.

1. Manage Student:
2. Manage Courses:
3. Manage Faculty:
4. Manage Research Groups
5. Manage Committees:

C.4 Observation

In this case study, it is observed that most of the changes are related to individuals. Thus, the change is usually attributed to the annotation triples which carry frequently evolving semantic information about the individuals. Some of the instance changes are attributed to addition of information about new faculty and students. In the case of adding new faculty, they create a web content about their research interest, the courses they teach, their publication, contact information, etc. When the students finish their studies, their information becomes inactive and later deleted from the web server.

The concepts in the ontology are relatively stable as compared to the frequent changes in the instances. Instructors add course content such as lecture notes, reference materials, demonstrations, guidelines, laboratory manuals, sample quizzes and exams. All these resources evolve frequently forcing the overall OCM system to evolve dynamically. However, it does not mean that the ontologies used in this domain are permanently stable. It only means, as compared to the changes in the content, the changes in the ontologies are rare.

This OCMS system is sensitive to changes that delete individuals. When individuals are deleted the information associated with them will be lost. Due to the nature of the information, any deletion which is not originally intended by the user creates a problem on the overall system. When concepts are changed, we further need to check whether there are dependent individuals associated with the changing entity. If there are dependent individuals, then we consider preserving the individuals before changing or deleting the entities.

In ontologies that focus on more on individuals, changing the $\mathcal{TB}$ox statements is preferable to amend inconsistencies. This means, to resolve inconsistency, we prefer changing the $\mathcal{TB}$ox definition than changing the assertion statements related to individuals. For example, if we find a staff who is a student, but if our $\mathcal{TB}$ox statement makes the staff and student concepts disjoint, to resolve the inconsistency, we prefer to delete the disjoint axiom (which is a $\mathcal{TB}$ox statement) than to delete the instance from either of the classes. In general, such OCMS are sensitive to changes in the $\mathcal{AB}$ox than the $\mathcal{TB}$ox statements.

C.5 Sample University Ontology

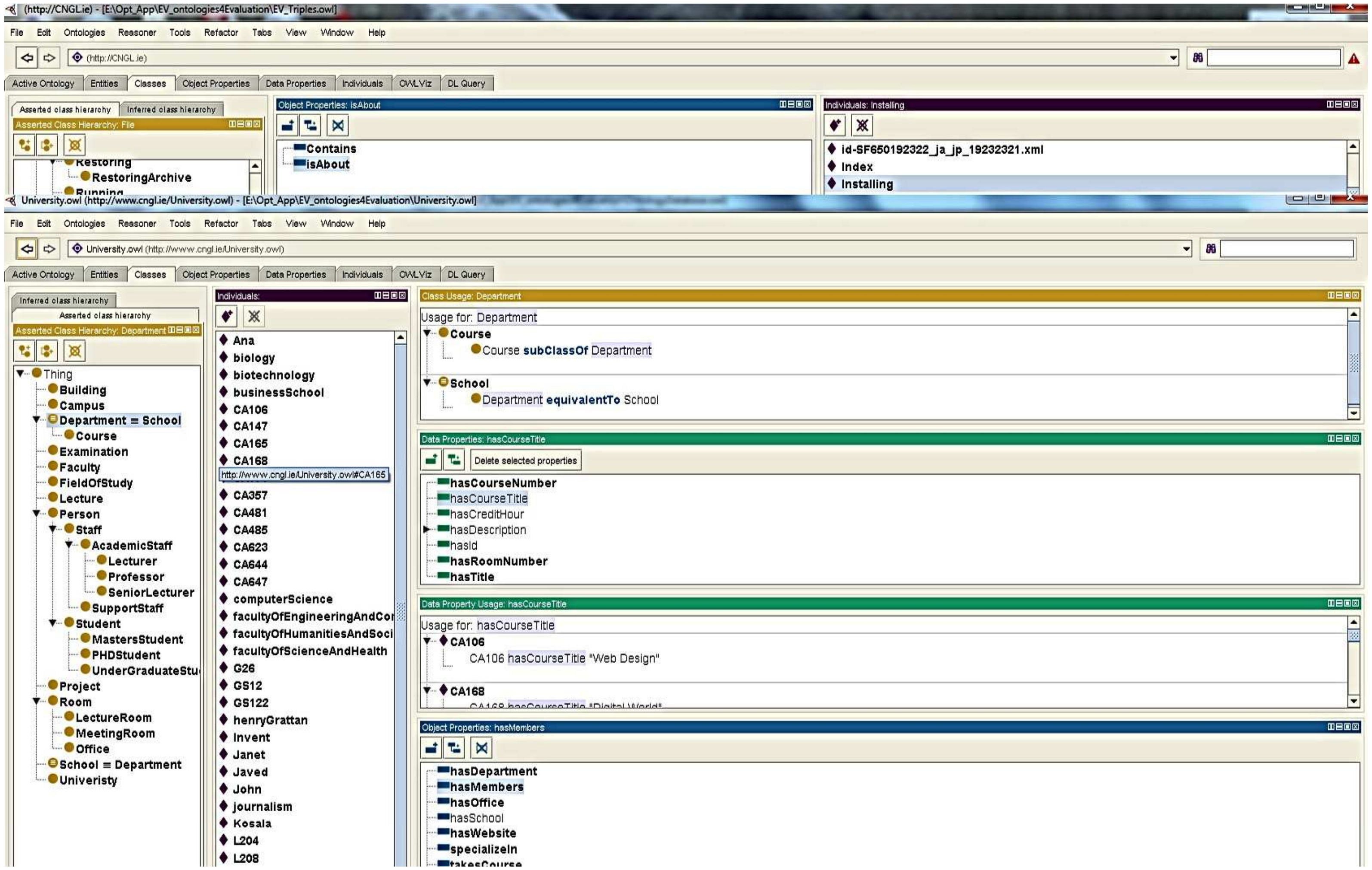

(http://CNGL.ie) - [E:\Opt_App\EV_ontologies4Evaluation\EV_Triples.owl]
File Edit Ontologies Reasoner Tools Refactor Tabs View Window Help
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query
Asserted class hierarchy Inferred class hierarchy
Asserted Class Hierarchy: File
Restoring
RestoringArchive
Running
Object Properties: isAbout
Contains
isAbout
Individuals: Installing
id-SF650192322_ja_jp_19232321.xml
Index
Installing
University.owl (http://www.cngl.ie/University.owl) - [E:\Opt_App\EV_ontologies4Evaluation\University.owl]
University.owl (http://www.cngl.ie/University.owl)
Inferred class hierarchy
Asserted class hierarchy
Asserted Class Hierarchy: Department
Thing
Building
Campus
Department ≡ School
Course
Examination
Faculty
FieldOfStudy
Lecture
Person
Staff
AcademicStaff
Lecturer
Professor
SeniorLecturer
SupportStaff
Student
MastersStudent
PHDStudent
UnderGraduateStu
Project
Room
LectureRoom
MeetingRoom
Office
School ≡ Department
Univeristy
Individuals:
Ana
biology
biotechnology
businessSchool
CA106
CA147
CA165
CA168
http://www.cngl.ie/University.owl#CA165
CA357
CA481
CA485
CA623
CA644
CA647
computerScience
facultyOfEngineeringAndCo
facultyOfHumanitiesAndSoci
facultyOfScienceAndHealth
G26
GS12
GS122
henryGrattan
Invent
Janet
Javed
John
journalism
Kosala
L204
L208
Class Usage: Department
Usage for: Department
Course
Course subClassOf Department
School
Department equivalentTo School
Data Properties: hasCourseTitle
Delete selected properties
hasCourseNumber
hasCourseTitle
hasCreditHour
hasDescription
hasId
hasRoomNumber
hasTitle
Data Property Usage: hasCourseTitle
Usage for: hasCourseTitle
CA106
CA106 hasCourseTitle "Web Design"
CA168
CA168 hasCourseTitle "Digital World"
Object Properties: hasMembers
hasDepartment
hasMembers
hasOffice
hasSchool
hasWebsite
specializeIn
takesCourse

Figure C.2: University ontology OCMS

Appendix D

Additional Analysis Results

This appendix contains additional results of the change impact analysis process. The results reflect different severity values and different weights of criteria for calculating cost of evolution.

D.1 Severity of Impacts

We take three cases where the severity of impacts is assigned. This assignment is based on the nature of the OCMS and the preference of the ontology engineer. The cases and their descriptions are discussed below.

D.1.1 Case 1

The first case collects severity values of impacts for the three case studies (Appendix 1 to 3) from expert ontology engineers. The collected severity values are used to calculate the average severity value that can serve as a default severity value. The average severity value is given in D.1.

D.1.2 Case 2

This case represents another severity value tuned to specific requirement. In this case, the integrity of the OCMS is crucial and the ontology engineers assign a high value to the severity values corresponding to the impacts related to integrity values. The severity values are presented in Table D.1

D.1.3 Case 3

This case represents a different severity value tuned to semantics of the OCMS than the integrity of the ontology. This case represents OCMS that are less affected by impacts that violate the integrity. Such OCMS give priority to the availability of description for the entities. The severity values of this case are given in Table D.1

Table D.1: Severity value assigned to case studies

No.	Semantic Impact	Acronym	Severity case 1	severity case 2	Severity case 3
1	Entity More described	(CMD,DPMD,OPMD,IMD)	15	50	25
2	Entity Less described	(CLD,DPLD,OPLD,ILD)	75	50	75
3	Entity More restricted	(OPMR)	75	50	75
4	Entity Less restricted	(OPLR)	35	50	25
5	Entity More expanded	(AME)	60	50	25
6	Entity Less expanded	(ALE)	80	50	50
7	Entity generalized	(CG,DPG,OPG,IG)	50	50	50
8	Entity specialized	(CS,DPS,OPS,IS)	70	50	50
9	Entity Incomparable	(CInc, DPInc, OPInc, IInc)	70	100	15
10	Unsatisfiable class/property	(UC,UDP,UOP)	100	100	15
11	Invalid instance/ instance property	(II, IIP)	80	100	15
No.	Structural Impact				
1	Orphan concepts	(OC)	80	100	10
2	Orphan Instance	(OI)	75	100	10
3	Property cyclic reference	(OPCR/DPCR)	90	90	10
4	Concept cyclic reference	(CCR)	95	90	10
5	Null reference to content layer	(NRC)	70	75	10
6	Null reference to ontology layer	(NRO)	70	75	10

D.2 Weight of Criteria

In Chapter 7 we identified four different scenarios where weight is assigned to the criteria. We further use those scenarios to combine them with the three cases identified above. Exploring the different combination of cases and scenarios is used to evaluate how the proposed system applies for different OCMS with different settings.

Table D.2: Different weights assigned for criteria

Scenario	Severity	Operation Type	Statement Type	Performance
Scenario 1	0.25	0.25	0.25	0.25
Scenario 2	0.70	0.10	0.10	0.10
Scenario 3	0.50	0.10	0.30	0.10
Scenario 4	0.30	0.30	0.20	0.20

D.3 Cost of Evolution for Different Settings

We analyse and identify the cost of evolution of OCMS systems by combining the three cases for severity setting and the four cases of weights of criteria. We have twelve different settings combining the cases with the scenarios. The cost is presented as follows.

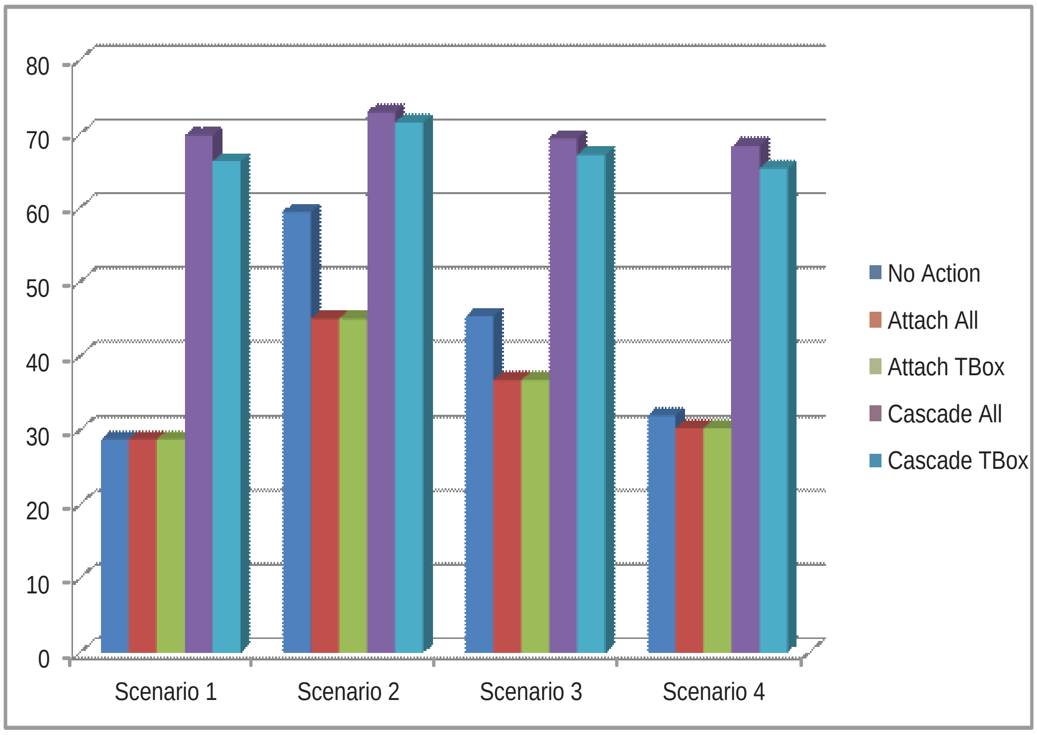


Figure D.1: Summary of cost of evolution - Case 1

Table D.3: Summary of cost of evolution - Case 1

Scenario	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
1	28.8	28.8	28.8	69.8	66.35
2	49.5	45.1	45.1	72.9	71.5
3	45.4	36.8	36.8	69.4	67.2
4	32.0	30.3	30.3	68.4	65.3

In the first case, the scenarios are compared based on the average severity values assigned in Table D.3. In this case no-action and attach strategies yield the same cost. But, when the weight assigned to the second scenario changes, favouring the severity of impacts (0.7), no-action strategy becomes costly, thus the preferable strategy is attach strategy (Attach all and attach $\mathcal{TB}$ox). Scenario 3 and Scenario 4 also yield a similar result but with a slight difference in the values.

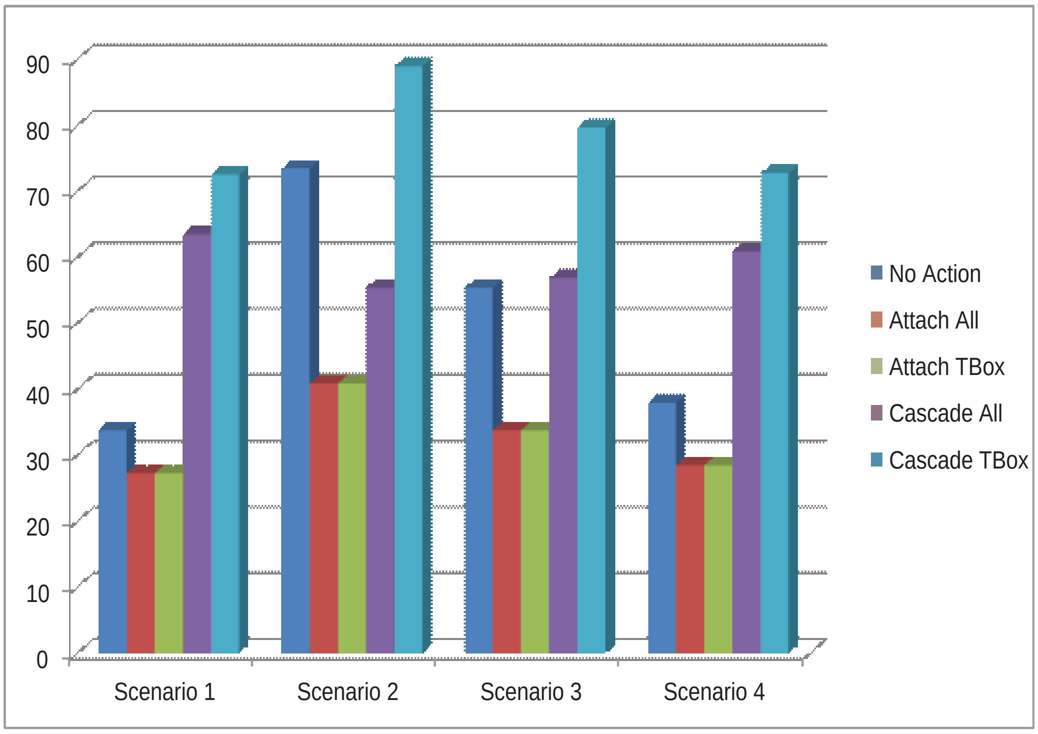


Figure D.2: Summary of cost of evolution - Case 2

Table D.4: Summary of cost of evolution - Case 2

Scenario	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
1	33.8	27.3	27.3	69.8	63.5
2	73.5	40.9	40.9	55.4	89.0
3	55.4	33.8	33.8	56.9	79.7
4	38.0	28.5	28.5	60.9	72.8

In the second case, the scenarios are compared based on the severity value assigned which gives a high severity value for impacts that affect integrity. In the first scenario, attach strategy (attach all and attach $\mathcal{TB}$ox) yields the optimal implementation option. When we look at scenario 2, the first strategy yields a higher value than the cascade strategy. Cascade strategy gives the second best value because; the severity of introducing orphans is higher than deleting the instances. The comparison between cascade all and cascade strategies also reflect this fact. When we cascade changes only to $\mathcal{TB}$ox statements, we leave the $\mathcal{AB}$ox statements unchanged and may introduce orphan instances. This makes the cost to go higher. In all the scenarios, the weight assignment does not make a difference on the best strategy selection.

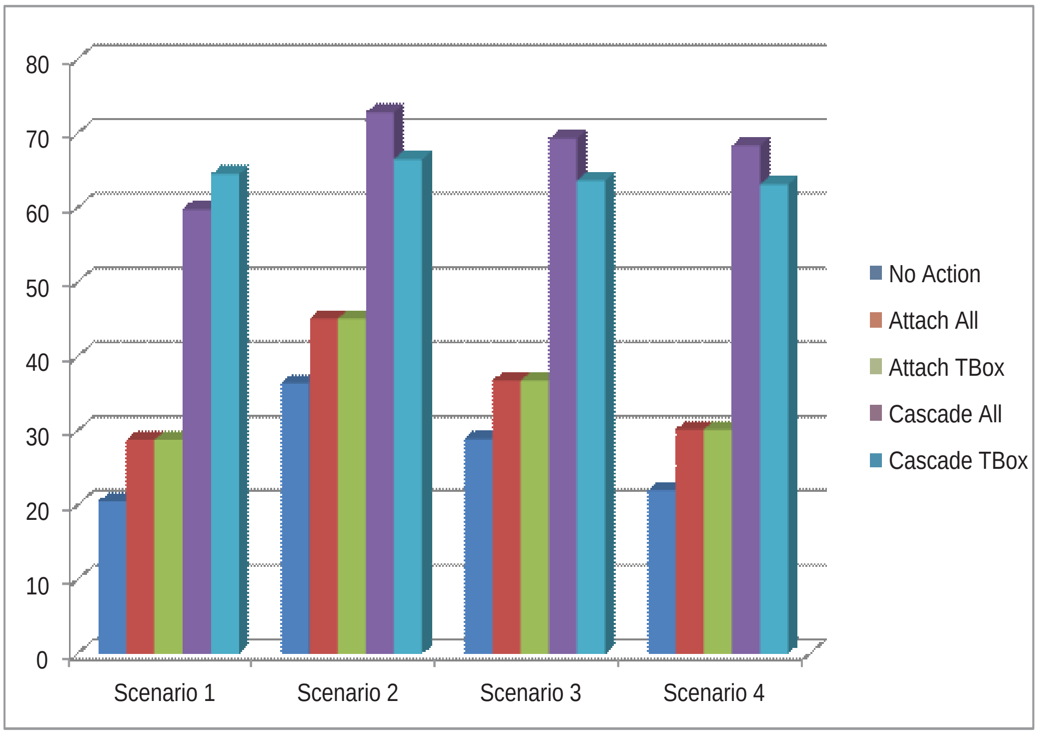


Figure D.3: Summary of cost of evolution - Case 3

Table D.5: Summary of cost of evolution - Case 3

Scenario	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5
1	20.5	28.8	28.8	59.8	64.6
2	36.4	45.1	45.1	72.9	66.6
3	28.9	36.8	36.8	69.4	63.7
4	22.0	30.2	30.2	68.4	63.2

In the third case, we give less attention to orphans, unsatisfiable classes etc. This setting yields a different result. In all the four scenarios, the best strategy is the no-action strategy which yields a minimum cost. This strategy is known for introducing orphans, and orphan instances. But, in situations where it is possible to allow orphan classes and instances (which is possible in OWL 2), this strategy is preferable over the other strategies. From this case one can conclude that trying to attach the orphan classes and instances to the parent classes is even costly. However, similar to the second case, the weight assigned to the criteria does not make a significance difference on the order of the optimal strategy, even if the cost of evolution is visibly different.

This might attribute to the high values of the severity criteria as compared to the other criteria. This happens because we use a small OCMS for the purpose of the experiment. The value of the other criteria will grow large if we use complex OCMS where large numbers of instances or classes are defined in it.

The analysis result clearly shows that the change impact analysis method and the optimal strategy selection depends on the severity values assigned to the impacts, the weights of the criteria and the size of the entities in a given OCMS. It further shows that the analysis is customizable and allows users to conduct a what-if analysis using different values and weights.

Appendix E

Questionnaire

This appendix contains questionnaire used to evaluate the different phases of the research. We present the whole evaluation setting and the questionnaires in the following sections.

E.1 Change Operations for Evaluation

Instruction

Load the respective ontologies using the system and implement the following change operations following the information provided for each stage of the evolution process.

Use University OCMS

1. Change scenario 1. Delete class (Student)
2. Change scenario 2 Add disjoint Class (Staff, Student)
3. Change scenario 3 Delete instance (John)

Use Ontology Database OCMS

1. Change scenario 4 Delete class (Table)
2. Change scenario 5 Add subclassOf (Schema, Relation_Schema)
3. Change scenario 6 Delete Object Property (hasSchema)

Use EV_Triples OCMS

1. Change scenario 7 Add Class (GUI)
2. Change scenario 8 Delete data Property (hasAverageSize)
3. . Change scenario 9 Delete instance (ID-123.xml)
4. Change scenario 10 Add instance (ID-1234.xml, File)

E.2 Questionnaires

Change impact analysis for Ontology-based Content management systems

Instructions:

- this questionnaire is to be filled after the attached change operations are implemented using the editor provided.
- Please complete the following question by putting a $\surd$ mark in the box [ ] next to your preferred answer.
- Please use the spaces available for writing your comments and observations.

1. The change representation allow me to choose between different implementation options for my original change request.

   [ ] Strongly Agree

   [ ] Agree

   [ ] Slightly Agree

   [ ] Slightly disagree

   [ ] Disagree

   [ ] Strongly Disagree

   Other ____

2. The system has provided all the change operations I need to implement the change requests.

   [ ] Strongly Agree

   [ ] Agree

   [ ] Slightly Agree

   [ ] Slightly disagree

   [ ] Disagree

   [ ] Strongly Disagree

   Other ____

3. The Change impact analysis identifies the impacts of my change request [ ] Strongly Agree

   [ ] Agree

   [ ] Slightly Agree

   [ ] Slightly disagree

   [ ] Disagree

   [ ] Strongly Disagree

   Please specify if there is an incorrect impact: ______________________________

4. The change impact analysis helps me better understand the effects of my change request

   [ ] Strongly Agree

   [ ] Agree

   [ ] Slightly Agree

   [ ] Slightly disagree

   [ ] Disagree

[ ] Strongly Disagree

Other ____

5. The change impact analysis identifies all the entities in the system that are affected

   [ ] Strongly Agree

   [ ] Agree

   [ ] Slightly Agree

   [ ] Slightly disagree

   [ ] Disagree

   [ ] Strongly Disagree

   Other ____

6. The change impact analysis correctly highlights the impacts of the change on the integrity of the OCMS (unsatisfiable classes and invalid instances)

   [ ] Strongly Agree

   [ ] Agree

   [ ] Slightly Agree

   [ ] Slightly disagree

   [ ] Disagree

   [ ] Strongly Disagree

   Other ____

7. The evolution process provides me the information related to my change request before I execute the change

   [ ] Strongly Agree

   [ ] Agree

   [ ] Slightly Agree

[ ] Slightly disagree

[ ] Disagree

[ ] Strongly Disagree

Other ____

8. The system helps me to find the optimal implementation strategy

   [ ] Strongly Agree

   [ ] Agree

   [ ] Slightly Agree

   [ ] Slightly disagree

   [ ] Disagree

   [ ] Strongly Disagree

   Other ____

9. The cost estimation is suitable to measure the impacts

   [ ] Strongly Agree

   [ ] Agree

   [ ] Slightly Agree

   [ ] Slightly disagree

   [ ] Disagree

   [ ] Strongly Disagree

   Other ____

10. I understand what I am doing at each step and understand the effects of my action during the evolution process.

    [ ] Strongly Agree

    [ ] Agree

[ ] Slightly Agree

[ ] Slightly disagree

[ ] Disagree

[ ] Strongly Disagree

Other ____