Change Impact Analysis and Optimization in Ontology-based Content Management Systems

Yalemisew Abgaz[a,1,*], Muhammad Javed[a], Claus Pahl[a,**]

[a] *Centre for Next Generation Localization (CNGL), School of Computing, Dublin City University, Dublin 9, Ireland*

Abstract

Ontologies are used to semantically enrich content in content management systems. Ontologies cover a wide range of disciplines enabling machines to understand meanings and reason in different contexts. We use ontologies for semantic annotation to facilitate understandability of the content by humans and machines. We call OCMS and ontologies evolve due to changes in the conceptualization, representation or specification of the domain knowledge. These changes are often substantial and frequent in relatively complex systems such as DBpedia[1]. Implementing the changes and adapting the OCMS accordingly requires a considerable effort. This is due to complex impacts of the changes on the ontologies, the content and dependent applications. We approach the problem of evolution by proposing a framework which clearly represents the dependencies of the components of an OCMS. We propose a layered OCMS framework which contains an ontology layer, content layer and annotation layer. Further, we present a novel approach for analysing impacts of atomic and composite change operations. The approach uses impact cancellation, impact balancing and impact transformation as a mechanism to analyse impacts of composite change operations. We propose a model which estimates the cost of evolving an OCMS using four criteria. The model ranks available evolution strategies and identifies the best strategy. The approach allows the ontology engineer to ex-ante evaluate the impacts and select an optimal strategy during the ontology evolution.

Keywords: Ontology Evolution, Change Impact Analysis, Content Management Systems, Optimizaiton

1. Introduction

Ontologies are used to semantically enrich content in content management systems. They are used to exchange semantics between humans, computers and other emerging devices [1]. In Ontology-based Content Management Systems (OCMS), ontologies facilitate a common understanding, specification, representation and interpretation of a shared knowledge between humans and machines [2]. This is achieved by embedding semantics by annotating the target content using ontologies. This allows both humans and computer systems to gain a common understanding about the target content.

In OCMS the content, the ontology and the annotation evolve frequently and dynamically. When new things are discovered, existing ones are deleted or modified, the respective content needs to be updated. Whenever there is a change in a domain, its conceptualization or specification, related ontologies need to evolve [3] [4]. When the meaning of the content changes, the annotation needs to evolve to respond to the change.

However, applying the changes and evolving the OCMS and its dependent systems is a challenging task for ontology engineers. During the evolution of an ontology, a change of one entity may cause many unseen and undesired changes and impacts on dependent entities [5] [6]. Identifying the changes and determining the impacts is a complex and time consuming activity which often causes inconsistencies in the ontology [7] [8], if it is not done correctly. Changes in the content also cause a change in the annotation. These changes further cause other unseen and undesired impacts on the instances. Thus, before permanent implementation of a change request, it is vital to conduct change impact analysis to understand the impacts and the impacted entities [9]. Ontology engineers benefit from the analysis in that it enables them to conduct what if analysis before permanent implementation of changes. It supports maintenance of ontologies and optimizes evolution of an OCMS. The analysis provides detailed and summarized impacts of changes in relatively large and complex OCMSs which consume much time and effort otherwise. In large ontology-based applications such as DBpedia[2] which contains more than 359 classes, 1,775 properties and 2,350,000 instances, the impact analysis will have a significant contribution for identifying impacts and for comparing and selecting optimal strategies.

Moreover, a given change request can be realized using different evolution strategies [5]. The evolution strategies differ both in the type and the number of change operations used to imple-

*Corresponding Author

**Principal corresponding Author

Email addresses: `yabgaz@computing.dcu.ie` (Yalemisew Abgaz), `mjaved@computing.dcu.ie` (Muhammad Javed), `cpahl@computing.dcu.ie` (Claus Pahl)

[2] http://wiki.dbpedia.org/Ontology

ment the requested change. 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. However, comparing and selecting the best strategy with an optimal solution is difficult, error prone and time consuming. Therefore, evolving and maintaining the overall integrity of an OCMS is a complex process. The evolution process involves the following core tasks.

- The representation of change requests using one or more available change realization options (evolution strategies).
- The characterization and analysis of impacts, impact types and the impacted entities.
- The estimation of the cost of evolution using selected criteria.
- The comparison of available evolution strategies and selection of the optimal strategy that ensures minimum cost of evolution.

While operational aspects of ontology evolution have been substantial in the past [5] [10] [11], this paper presents a novel approach to impact analysis that assist transparent, consistent and optimized evolution of an OCMS. To this end, we present a bottom-up change impact analysis process. Following this, we propose a model that analyses and selects an optimal implementation strategy. The change impact analysis approach uses novel techniques such as individual change impact analysis and composite change impact analysis which employs impact cancellation, impact balancing and impact transformation as a mechanism to analyse impacts of composite change operations. We extend existing research by incorporating impacts of change operations, analysing the severity of impacts, defining criteria for estimating the cost of evolving an OCMS and proposing a model for selection of optimal implementation strategy. The contribution of this paper includes the following.

- realization of change requests using existing and customized evolution strategies (Section 4.1 to 4.2).
- analysis of impacts of atomic and composite change operations. The approach provides useful information beyond what is added or deleted. The proposed method ensures the evolution of an OCMS in a consistent, accurate and transparent manner (Section 5.1 to 5.3).
- quantitative estimation of the impacts of the change operations, and the cost of evolution (Section 6.1 to6.5).
- selection of an optimal implementation that ensures minimum cost of evolution (Section 6.6 to 7).

This paper is the extension of previous publications on change impact analysis [12] [13] [14] [15] and [16].

2. Related Work

In recent years, several researchers addressed the problem of changes in ontologies and ontology-based applications.

A six-phase evolution framework is widely used to evolve ontologies [17] [18] [19]. The framework [17] focuses on change detection, change representation, semantics of change, change propagation, change validation and change implementation. The semantics of change phase deals with derived change operations to consistently evolve the ontology. This phase focuses on the effects of the changes on dependent entities. Our work provides detailed analysis of the change operations and present how and why the changes affect the entities.

Validity of data instances against ontology evolution is suggested by [8]. The work discusses evolution of OWL ontologies with the aim of guaranteeing validity of data instances. The work distinguishes structural and semantic changes and identifies instances that are invalidated by any of these changes. This work specifically focuses on $\mathcal{AB}ox$ statements. Our approach extends this work to analyse the structural and semantic impacts that arise from the changes.

The authors [7] [20] [21] have proposed a formal approach for RDF/s ontology evolution. 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 in relation to the validity rules presented by the authors. The authors in [7] incorporate minimal change criterion to ensure minimum number of changes to evolve ontologies. The work in belief change principle 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, impacts and sensitivity of the ontology to $\mathcal{AB}$ox or $\mathcal{TB}$ox statements

Authors in [22] developed PromptDiff to compare different versions of an ontology. It compares two versions of an ontology and analyse the differences in terms of additions and deletions. 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, [23] [24] developed a successor of PromptDiff that use the heuristics used in PromptDiff to support OWL2 ontologies. The authors [24] have suggested 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 in [25] have presented a pluggable difference engine which aligns ontology entities before conducting comparison. The difference engine highlights additions, removals and renamings 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 work view changes with the differences between two versions and does not deal with the change operations that causes the differences and their impacts either individually or as a composite change. It further requires the original ontology and the evolved ontology after the changes are implemented. However, support for analysis of impacts of the changes before implementation is not available in all Diff approaches.

Content CVS [26] [27] allows the concurrent development of ontologies following concurrent versioning methods in software development. Concurrent changes, version comparison, conflict detection and resolution and version merging are included in the Content CVS system. When ontologies evolve conflicting changes occur and result inconsistency. The system keeps local and global versions and compares the ontologies to detect conflicting changes using structural and semantic differences. Conflicting changes or unintended entailments are presented to the user with zero or more resolution plans. The authors in [28] have proposed a web-based tool, CODEX (COmplex Ontology Diff EXplorer). The tool computes ontology differences and presents the number of changes, Diff sizes and the growth rates of changes. It allows exploration of elements that have been influenced by the changes. Codex uses Diffs to analyse impacts of change operations. This work is closely related to our work; however we view impacts as results of individual change operations implemented in the original ontology than the difference between two ontology versions. Change impacts are attributed to the atomic and composite changes that are implemented on the original ontology and an ex-ante evaluation of impact analysis and selection of strategies based on the analysis of impacts is suggested.

Optimization of ontology evolution and optimal selection of evolution strategies is given a little attention in the state-of-the-art literature. The authors in [29] conduct a study on user defined ontology changes 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. This research explores a new area in evolution of ontology-based content management by covering optimal strategy selection using quantitative analysis of parameters.

In this paper, we extend the evolution strategies suggested in [5] to generate change operations to implement the requested changes. We exploited structural and semantic changes suggested in [8] to identify structural and semantic impacts. The bottom-up change impact analysis approach from the perspective of the change operations and an optimal implementation which focuses on criteria such as severity of impacts, statement types, operation types and number of change operations are the novel contributions of this work. This work mainly focuses on the impacts of change operations on asserted axioms of ontologies and annotations. Our next stage will be incorporating the impacts of change operations on inferred axioms in the OCMS.

3. OCMS Principles

OCMS are systems that use ontologies to semantically enrich content. OCMS are used to facilitate accessibility of content for both humans and machines by integrating semantics about the content using ontologies. An OCMS is composed of three different layers. The first layer is the ontology layer (represented using OWL2), the second is the annotation layer (represented using RDF triples) and the third one is the content layer (set of documents). The layered architecture is presented in (Fig. 1).

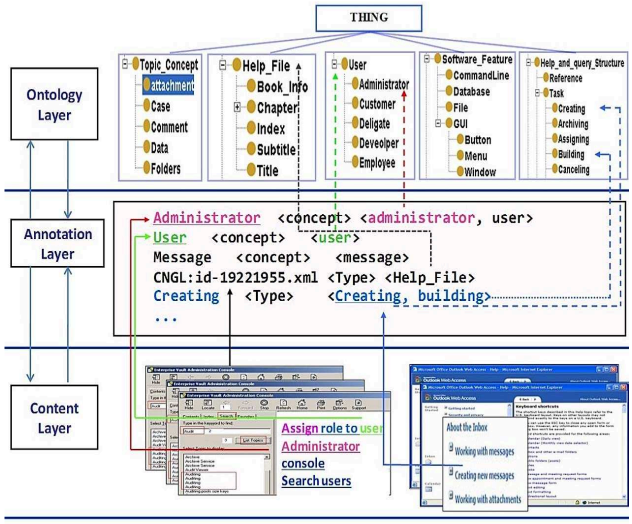

Figure 1: Layered architecture of OCMS

Ontology Layer. This layer provides the specification of a shared conceptualization of a domain. This means ontologies provide a common ground for understanding, conceptualization, representation and interpretation of domain concepts uniformly across different systems, languages and formats. They provide a representation of knowledge that allows machines to reason about known facts and generate new knowledge from them.

Content Layer. Content, in this paper, refers to any digital information that is in a textual format that includes structured or semi-structured documents, web pages, executable content, software help files etc. [6][12]. An OCMS essentially deals with content in the form of books, web pages, blogs, newspapers, software products, documentations, help files, reports, publications, etc. [6].

Annotation Layer. Annotation is a process of linking content with ontology entities to provide better semantics to the content. The aim of semantic annotation is to explicitly identify concepts and relationships between concepts in the content [30]. 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.

3.1. Running Example

In this section, we present a Software Help Management System (SWHMS) case study as a running example to illustrate the process and to validate the proposed solution later. The SWHMS is designed to support users of an enterprise software product to efficiently exploit the help files that are distributed with the software or available online. The target software product provides integrated content archiving that enables users to store, manage and discover organizational information. The case study focuses on enhancing the overall help management system using various domain ontologies by semantically annotating the help files and the software product. The help files describe the components, purpose of the software and the tasks, procedures, steps, etc., required to use the software and to troubleshoot problems. The help files are prepared by domain experts in software and digital archiving.

In this case study, the help files are extracted from different version of the software. The help files are either semi-structured or structured files in HTML and XML formats respectively. The files are organized in different folders using a conceptual structure of the software. For each version concept maps which describe the relationship between concepts discussed in the help files are available. For this study, we gained access to two versions of the software help files. The first version contains 162 HTML files organized into 4 folders which represent the four components of the software. The second version contains 839 XML files organized in 17 folders. These folders are used to categorize the help files based on the available software components. We built four primary ontologies for supporting SWHMS. A high-level description of these ontologies and their dependencies are depicted in Figure 2.

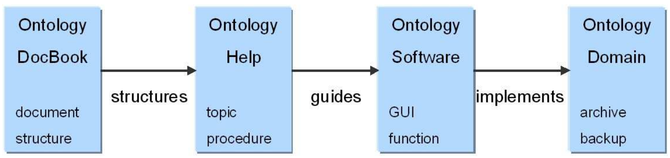

Figure 2: A high level view of software help management ontology

The first ontology is the DocBook ontology which describes the overall structure of the help files [31]. The DocBook ontology is constructed by extracting the structural entities from the available DocBook files [32] [33]. 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 help ontology guides the software ontology in a way that explains usability of the topics, procedures, steps, etc. by the software ontology. The software 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 the software at hand. The domain ontology is also known as the application ontology. In our case study the domain of the software is digital archiving which includes backup, searching, sharing, etc.

The rationale behind selecting this case study is that it covers a wide range of topics from help to software systems domains. In this software product, the help files are directly associated to the various domains. Help files are organized using document structures, software concepts such as GUI elements, commands, hardware and software requirements, etc. Moreover, since concepts and instances are distributed throughout the help files, they create a strong link between the instances in the content and the concepts in the ontologies. This makes it of great interest to investigate impacts of concept changes and instance changes because the changes made in the content of the help files will have an impact on the ontology and vice versa.

3.2. Graph-based Representation of OCMS

We use a graph-based formalism to represent the OCMS. We choose graph-based formalization for the following basic reasons. First, graphs provide exhaustive theory support and reduce the problem to a well-studied topics in graph theory [34]. Graphs have some proven efficiency for searching subgraphs, nodes and edges [35]. Second, graphs provide an appropriate data structure to model ontologies represented as RDF and OWL [36] [37]. Finally, graphs visualize complex data in a simple and understandable way.

An OCMS is represented as a 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 3.

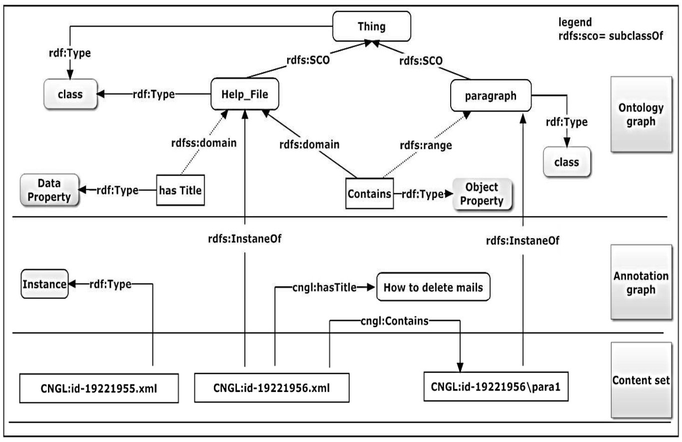

Figure 3: Graph-based representation of OCMSs

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 [38]. 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 $\alpha \in CA \cup DPA \cup OPA \cup IA \cup RA$. The representation follows the OWL2 specification[3].

In general, we treat properties as nodes and property instances as edges. When we define properties as part of an ontology, we represent them as nodes and when we use properties for annotation, we represent them as edges of the annotation graph. This means, we represent the properties as a node and link them with other class or property nodes in the ontology graph. We represent property instances as edges that link two instances in the annotation graph.

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

[3]http://www.w3.org/TR/owl2-quick-reference/

CA= {*subClassOf, disjointClasses, equivalentClasses*}
DPA= {*subDataPropertyOf, dataPropertyRange, dataProperty Domain, disjointDataProperties, equivalentDataProperties, functionalDataProperty*}
OPA= {*subObjectPropertyOf, objectPropertyRange, objectProperty Domain, disjointObjectProperties, inverseObjectProperties, equivalentObjectProperties, symmerticObjectProperties, functionalObjectProperty, inverseFunctionalObjectProperties, transitiveObjectProperty, reflexiveObjectProperty, irreflexiveObjectProperty*}
IA= {*sameindividual, differentIndividuals, classAssertion, DataPropertyAssertion, objectPropertyAssertion* }
RA= {*allValuesFrom, someValuesFrom, minimumcardinality, maximumcardinality, exactCardinality*}

semi-structured document or elements of a document. In the content layer, such content is represented as a node. 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 URI.

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. We represent documents as nodes in the OCMS graph.

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 a function $label(e)$. The ***label*** of a node n is the URI associated with the node and is given by a function $label(n)$. All the edges of a node n are given by a function $edges(n)$. It returns all the edges as (n, α, m) where n is the target node and m is any node linked to n via α.

3.3. 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 additional change operations to complete the requested change. 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 4 outlines the phases of the change impact analysis framework and their interactions.

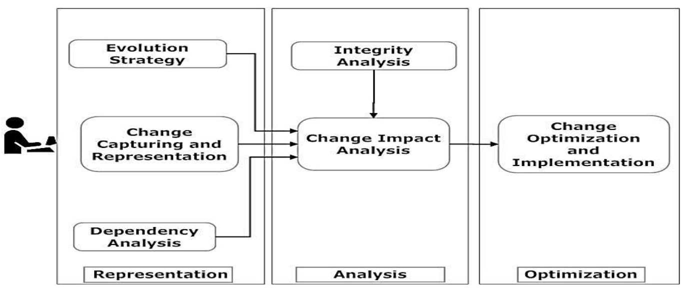

Figure 4: The change impact analysis framework

4. Change Request Capturing and Representation

The objective of this phase is to represent detected changes using suitable change operations that ensure 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 [5] [39]. A requested change is implemented either by an atomic change operation which does a unit task or a composite change operation composed of two or more atomic or composite changes. 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 [40] [41]. Change representation uses the outputs of the evolution strategies and the dependency analysis. To fully implement a requested change we may generate complementary change operations. The requested change operation and the generated change operations make the complete change operation. Complementary changes are generated using dependencies and evolution strategies. Dependency analysis is discussed in Section 4.1 and evolution strategies are discussed in Section 4.2.

4.1. Dependency Analysis for Change Representation

Before we present the details of change impact analysis, we discuss dependencies that are crucial inputs 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. 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. 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 [42]. Characterization, representation and analysis of dependencies within and among the ontology, the annotation and the content layers are important. 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.

4.1.1. General Properties of Dependency

A dependency can be direct or indirect. 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 $indDep(N_1, N_3), if\ \exists N_2.\ Dep(N_1, N_2) \wedge Dep(N_2, N_3) \wedge N_1 \neq N_2 \neq N_3$. A direct dependency does not require intermediate dependency between entities. Algorithm 1 in general returns all the direct dependent entities of a given entity, in this specific case a given class. This means all classes that are directly connected to a given class are returned. 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. 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 $\exists N_2.\ Dep(N_1, N_2)$

$\wedge\ \neg\exists N_3.\ Dep(N_1, N_3) \wedge (N_2 \neq N_3)$. Algorithm 2 identifies all total dependent entities of a given entity. It identifies all total dependent entities whether they are directly or indirectly dependent. 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 $PDep = Dep - TDep$. Algorithm 2 can be customized to return partial dependent entities by changing the return value.

Algorithm 1 getDirectDependentClasses(G,c)

Input: Graph G, Class node c
Output: direct dependent classes (d)
$d \leftarrow \emptyset$
if the node c exists in G **then**
    for each edge $E_i = (m, \alpha, c)$ directed to c **do**
        if $label(E_i) =$ "$subClassOf$" $\wedge\ type(m) =$ "$class$" **then**
            add m to d
        end if
    end for
end if
return d

Algorithm 2 getTotalDependentClasses(G,c)

Input : Graph G, Class node c
Output: all total dependent classes=d
$d \leftarrow \emptyset$, contained=true
Set depCls=$\emptyset$,totalDepCls=$\emptyset$,partialDepCls=$\emptyset$, super=$\emptyset$
depCls$\leftarrow$ getAllDependentClasses(G,c)
for each concept c_i in depCls **do**
    if count(getSuperClasses(c_i)=1 **then**
        super $\leftarrow$ getSuperClasses(G, c_i)
        if super not in partialDepcls **then**
            add c_i to totalDepCls
        end if
    else
        super $\leftarrow$ getSuperClasses(G, c_i)
        contained=true
        for each sc in super **do**
            if sc not in depCls **then**
                contained=false
            end if
        end for
    end if
    if contained=true **then**
        add c_i to totalDepCls
    else
        add c_i to partialDepCls
    end if
end for
return totalDepCls

4.1.2. Types of Dependency

We categorize dependencies as dependencies within layers and across layers based on the layers the entities come from. Using an empirical study, the following dependency types 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 the OCMS graph $G = (N, E)$ and the examples are taken from Figure 3.

Dependency within a Layer. These dependencies occur between entities that come from the same layer.

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 $\exists C_2.\ Dep(C_1, C_2) \wedge (label(E_i = (C_1, \alpha, C_2)) =$ "$subClassOf$"$) \wedge (type(C_1) = type(C_2) =$ "$class$"$)$. 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 $(E_i = (C_1, \alpha, N_i) \vee E_i = (N_i, \alpha, C_1)) \wedge (type(C_1) = type(N_i) =$ "$class$"$)$. 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 $E_i = (N_i, \alpha, C_1) \wedge (type(C_1) =$ "$class$" $\wedge\ \alpha \in RA)$. 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 $\exists P_2.\ Dep(P_1, P_2) \wedge (label(E_i = (P_1, \alpha, P_2)) =$ "$subPropertyOf$"$) \wedge (type(P_1) = type(P_2) =$ "$property$"$)$. 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 $E_i = (P_1, \alpha, N_i) \vee E_i = (N_i, \alpha, P_1) \wedge (type(P_1) =$ "$property$"$)$.
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 $E_i = (N_1, \alpha, P_1) \vee E_i = (P_1, \alpha, N_1) \wedge (type(P_1) =$ "$property$"$)$.
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 $E_i = (C_1, \alpha, N_i) \wedge (label(E_i) =$ "$subClassOf$"$) \wedge (type(N_1) =$ "$class$"$)$. 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.

Dependency across Layers. These dependencies occur across entities in the three layers. Content-annotation dependency and ontology-annotation dependency are the main dependency types. We identify the following dependencies which exist across layers.

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 $E_i = (I_1, \alpha, C_1) \wedge (label(E_i) =$ "*classAssertion*") $\wedge$ ($type(I_1)$ = "*individual*") $\wedge$ ($type(C_1)$ = "*class*"). 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 $(label(E_i) = P_1) \wedge (type(N_i) =$ "*instance*") $\vee$ ($type(N_j)$ = "*instance*"). 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 $(E_i = (I_1, \alpha, N_i) \vee E_i = (N_i, \alpha, I_1)) \wedge (type(I_1) = type(N_i)$ = "*instance*").
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 $E_i = (I_1, \alpha, N_2) \wedge$ ($label(E_i)$ = "$instanceOf$") $\wedge$ ($type(i_1)$ = "*instance*").

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 result in the propagation of impacts in the OCMS. Using these dependencies, we developed algorithms to identify dependent entities.

4.2. Evolution Strategies

The selection of an optimal evolution depends on the different options available to realize a change request. A change request can be realized using different ways called evolution strategies. These evolution 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 constraints [43] [5] [44].

We identified four different strategies used by existing systems [5] 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. The details of each of the techniques used by the change implementation strategies are discussed below.

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 complete change operation does not include any change operation other than the ones that remove the traces of the target entity from the OCMS.

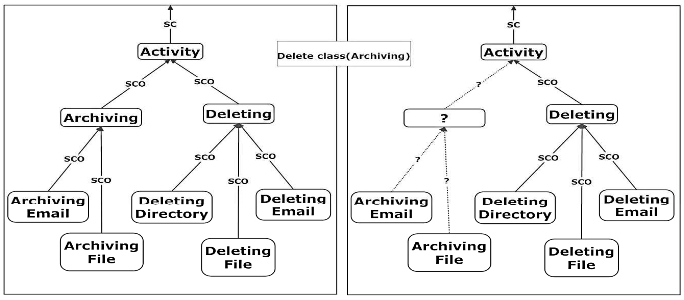

Figure 5: No-Action Strategy

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

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

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 entity, we need to change all its dependent entities. In case of deletion, when we delete an entity, the deletion propagates to all its total 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.

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

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

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:

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

Using this strategy, for example, Delete Class (*Archiving*) in (Figure 5) 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.

The Cascade and Attach strategies can be further customized to Attach only TBox statements. This means the customized strategies implement the changes only for TBox statements without considering ABox statements.

5. Change Impact Analysis Process

The change impact analysis process (5) includes analysis of impacts and their implication on the integrity of the OCMS. In this section we will focus on identifying and characterizing impacts, causes of impacts and the nature of the impacts.

5.1. 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 impact preconditions at which the impacts occur. an impact precondition defines the necessary condition that needs to be satisfied for the impact to 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 [19] [45] [8]. 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.

$$
\begin{aligned}
&Imp : (ACh, P) \rightarrow (Impact) \text{ where:} \\
&\quad Impact = StrImp \cup SemImp \\
&\quad ACh = \text{Atomic change}, P = \text{precondition}
\end{aligned}
$$

The structural and semantic impacts of an atomic change operation are discussed in the following subsections.

5.1.1. Structural Impacts

Structural impacts are impacts that change the *structural dependency* 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 [5].

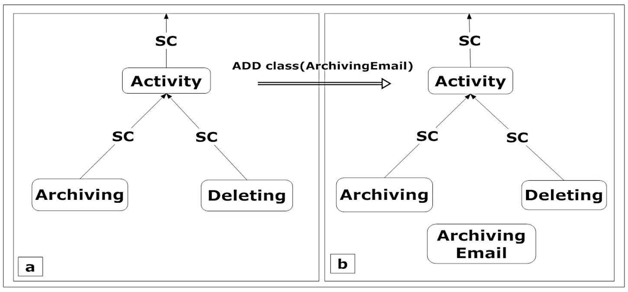

Figure 6: Example of structural impact

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

There are two types of structural impacts. The first type focuses on structural impacts that cause structural integrity violations (e.g. addition of cyclic reference). We call these impacts integrity-violating impacts. The second type focuses on changes, which are results or consequences of a given action (e.g. addition of an entity). These are caused by changes that add or remove entities. We call these impacts integrity non-violating impacts.

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.a), we can see that there are three entities. Due to a change operation *Add Class*(*ArchivingEmail*), the OCMS evolves to the second version (Figure 6.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.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 *Add Class*(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 following structural impacts of atomic change operations are either integrity violating or integrity non-violating depending on the impact preconditions.

- **Orphan Concept (*OC*)** occurs when a given concept 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 concept node c_i in N, excluding the root concept of the ontology, should have at least one super concept c in N, giving closure to $c_i : \forall\ c_i \in N \setminus \{Root\} \wedge type(c_i) = \text{“}Class\text{”} \rightarrow \exists c \in N.\ CCDep(c_i, c)$.

- **Concept Cyclic Reference (*CCR*)** occurs when a change operation introduces a cyclic reference to concepts. It violates the concept hierarchy invariant. The concept 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 concept 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 concept 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.

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.

5.1.2. 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 [8] and derived semantic impacts from the changes. The semantic impact of an atomic change operation is defined as:

$SemImp(ACh) = \{EG, ES, EMD, ELD, OPMR, OPLR, AME, ALE, UE, IE\}$
where:

- **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.

- **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 [46].

- **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 [46].

- **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 [47].

- **Invalid Entity (**IE**)** occurs when a change on a given instance or instance property introduces invalid interpretation [8].

Researchers [5] [8] have categorized 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.

Semantic impacts are caused by structural changes [8]. Some of the structural changes, which involve axioms that specify relationships between concepts (subclass of, intersectionOf, disjointWith, complementOf) and relations between properties and concepts (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.

5.2. 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. To do this, we define all the potential structural and semantic impacts of atomic change operations together with the affected entities and the impact preconditions (Table 2).

If the change operation has a precondition for a given impact, we will check if the precondition is satisfied. If the precondition is satisfied, we will take the impact and the target entity as an impact of the change operation, otherwise ignore that impact. For semantic impacts that cause unsatisfiability or inconsistency, the individual change operation may not be the sole reason. In such situations, we further inspect other change operations to explain the reason for the violation of the integrity and to resolve 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.

5.2.1. Impacts of Atomic Change Operations

We identified different atomic change operations and studied their semantic and structural impacts. To discuss atomic change impact analysis, we take frequently observed [48] change operations. The partial list of the impacts of atomic change operations and their preconditions is given in Table 1.

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 once-off 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 composition 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.

A general algorithm that attaches the structural and the semantic impacts of change operations is given in Algorithm 3. The algorithm takes the complete change operation, analyses the impacts of the atomic change operations and returns the associated impacts of the change operations. Any ontology evo-

Table 1: 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	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
11	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)
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)	disjointDataProperty$(dp, dq) \wedge \exists dp_1$. **PPDep**$(dp_1, dq)$
			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)	DisjointClasses$(c, d) \wedge \exists c$. **CCDep**$(c_1, c) \wedge$ **CCDep**(c_2, d)
			$CMD(c_1), CMD(c_2)$	None
			II(I)	$\exists i$. **CIDep**$(i, c_1) \wedge$ **CIDep**(i, c_2)

Algorithm 3 **Assign Individual Change Impacts** ($CCh, Impact$)

```
Input : Complete Change operation (CCh), Change impacts(Impact)
Output: Complete Change operation with impacts
for each atomic change operation(ACh) in CCh do
    if ACh is found in change impacts then
        assign corresponding impact to Imp
        for each strImp in Imp do
            if structural precondition(imp)=true then
                attach the affected entity to the strImp
                attach strImp to ACh
            end if
        end for
        for each semImp in Imp do
            if semantic precondition(imp)=true then
                attach the affected entity to the semImp
                attach semImp to ACh
            end if
        end for
    end if
end for
return CCh
```

lution 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 aggregation of the impacts of the atomic change operations. Thus, we require a different impact analysis strategy at the composite level.

5.3. Composite Change Impact Analysis

Composite change impact analysis focuses on analysing impacts of two or more change operations when they are executed together. Atomic change impact analysis shows only the impacts of that specific atomic change operation. When we implement a requested change, we often have more than one atomic change operation to fully implement the requested change. 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 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, impact balancing and impact transformation that exploit dependencies between individual changes and impacts. These approaches use rules and optimizes the result by removing redundant impacts.

5.3.1. 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 occurs between a pair of addition or a pair of deletion operations. 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.

Impact cancellation uses the following rules to identify and cancel impacts of composite change operations.

- **Rule 1.** When a target entity is affected by an operation ACh_1, and if that target entity is deleted by another operation ACh_2, the applicable structural and semantic impacts of ACh_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. Table 2 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 2: 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.

For example, the impact of *Delete SubClass (DeletingFile, Deleting)* and *Delete Class (Deleting)* is given in Table 3. 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 3: 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

When 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.

In Table 4 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 links 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.

Table 4: 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)

5.3.2. 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. Candidate impacts for balancing is presented in Table 5.

  $Imp\{ACh_1, ACh_2\} = \emptyset$ 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 5: 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)

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.

After balancing of the change operations in Table 6, 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 high level change impact, which is

Table 6: 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 SubClassOf (DeletingFile, Deleting)*	None	*CLD(Deleting)*

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 impact. In such situations, we move to the impact transformation step.

5.3.3. 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 introduced 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. For $ACh_1 = AddSubclassOf(x, y)$ and $ACh_2 = DeleteSubclassOf(x, y')$

  $Imp\{ACh_1, ACh_2\}$ =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 $y \not\subset y' \wedge y' \not\subset y$

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

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 Table 7, the first semantic 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 concept *DeletingFile* goes up in the hierarchy.

Table 7: 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 balancing		
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. After applying composite impact analysis on individual change impacts, we achieve 11.4%, 21.2% and 20.7% reduction of impacts for No-Action, Cascade and Attach strategies respectively. This shows that composite change impact analysis is capable of filtering redundant impacts and presenting the remaining impacts precisely.

6. Optimal Strategy Selection and Implementation

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 to select an optimal strategy that implements a requested change. We further introduce an optimization framework that 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 available implementation strategies and the impacts of the complete change operations. Each strategy is evaluated using four criteria that serve

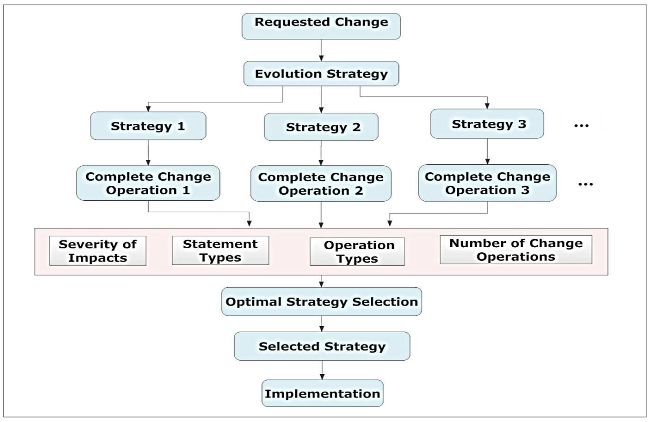

Figure 7: Framework for selecting optimal strategy

as an input for calculating cost of evolution. Severity of impacts, operation types, statement types and performance are the criteria used in the framework. The cost of evolution is calculated using quantitative measures of the criteria. According to the nature of the OCMS, ontology engineers assign weights to each of the criteria to show the significance of one criterion over another.

6.1. Severity of Impacts

In the previous section we identified structural and semantic impacts. We observed that some of these impacts are severe and cause more problems than the others. Thus, it is 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 severity, which makes the OCMS inconsistent. 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. Setting severity value of impacts in a given OCMS depends on the requirements defined by the ontology engineer or the content manager. We use heuristics to measure the severity value of the impacts. The heuristics considers 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.

We use the following settings of severity in this experiment. For semantic impacts, $\{(EMD,15), (ELD,75), (OPMR,75), (OPLR,35), (AME,60), (ALE,80), (EG,50), (ES,70), (UC,100), (UDDP,100), (UOP,100), (II,80), (IIP,80)\}$. For structural impacts, severity is set to $\{(OC,80), (OI,75), (OPCR,90), (DPCR,90), (CCR,95), (NRC,70), (NRO,70)\}$. These severity values are average severity values assigned by experts to the OCMS discussed in the running example.

Once a severity value is assigned to structural and semantic impacts, calculating the severity of impacts caused by the change operations in a given strategy is done by defining threshold, maximum and average severity values. We analyse the severity of the impacts after the composite change impact analysis is performed.

Severity Threshold. 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 cut-off 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, the maximum severity value will be taken as a representative value for that specific strategy [49] [50] [51]. 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.

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

For example, if a threshold is set to be T=80 and if one strategy has severity values of impacts as $\{75, 15, 35\}$, then S=Avg(s)= 41.6. If another strategy has severity values as $\{100, 15, 100\}$, then S=Max(s)=100.

The average severity is calculated as follows. Here f_i represents the frequency of s_i

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

We follow this approach to reduce the effects of frequent but less severe impacts on the overall estimation of severity. By definition, all impacts above the severity threshold should be avoided by any means. Anything which is less than the threshold is represented by the average severity. The higher value of the severity indicates a lesser desirability of the solution.

6.2. Type of Change Operation

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 complete 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 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, we conduct dependency analysis and cascade the change to all dependent entities. In terms of time and resource, a deletion operation incurs extra cost compared to the addition operation.

Whenever there is a difference of performance between addition and deletion operations, we assign a different weight to the change operations [50] [51]. 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 is, the higher the desirability of the change operation. Thus, for a given complete 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.

$$WF(A) = W(A) * |A|$$

$$WF(D) = W(D) * |D|$$

$$OT = WF(A) + WF(D)$$

Where:
OT= Operation Type
WF(A) is weighted frequency of Additions
WF(D) is weighted frequency of Deletions
$0 \le W(A) \le 1$, $0 \le W(D) \le 1$ and $W(A)$+ $W(D) = 1$
$|A|$=number of additions and $|D|$= number of deletions

6.3. Statement Types

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 a university administration OCMS with more instances than concepts, 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 case of university administration OCMS. Statement type serves as a means of selecting an optimal implementation strategy. This criterion corresponds to the OWL profiles. OWL profiles distinguish between ontologies heavily used for instance annotation (OWL-QL[4]) and ontologies used for logical expressions (OWL-EL[5]). Ontologies adhering to OWL-QL are more sensitive to $\mathcal{AB}$ox statements and ontologies adhering to OWL-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 complete 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.

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

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

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

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
$0 \le W(\mathcal{AB}ox) \le 1 \wedge 0 \le W(\mathcal{TB}ox) \le 1$
$|\mathcal{AB}ox|$= number of $\mathcal{AB}$ox statements,
$|\mathcal{TB}ox|$= number of $\mathcal{TB}$ox statements

6.4. Performance of Change Operations

We measure performance using the number of atomic change operation required to implement the change. This is done by counting the number of atomic change operations in the complete change operation. The rationale for using this criterion is that the number of atomic change operations affects the performance and could be used to distinguish between two available implementation strategies.

$$P = |ACh|$$

Where:
$ACh \in CCh$
P= Performance
$|ACh|$ = number of atomic change operations
$|CCh|$ = number of composite change operations

6.5. Cost of Evolution

Measuring the cost of evolution to select the optimal strategy based on the impact analysis is the central process of the implementation phase. 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. However, 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 may give more weight to the statement types and ignore additions and deletions. Thus, we need to compare each of the strategies individually to select the optimal strategy for the given criteria at hand. However, a single criterion does not fully characterize the available strategy. 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 $\{w_1, w_2, w_3, w_4\}$ for all criteria based on their importance in a given OCMS. 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.

[4]http://www.w3.org/TR/owl2-profiles/#OWL_2_QL
[5]http://www.w3.org/TR/owl2-profiles/#OWL_2_EL

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

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. This approach further allows the ontology engineer to remove one or more criteria if that criterion is not useful in that OCMS. This can be done by assigning a value zero for the weight of the corresponding criterion.

6.6. Optimal Strategy Selection

The optimal strategy selection exploits the cost of evolution for finding the optimal implementation strategy. The selection of the best strategy is based on the selection of a strategy with a minimum cost.

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

A strategy with a minimum cost implies that the strategy will evolve the OCMS with minimum severity of impacts, minimum number of change operations, minimum number of preferred statement types and operation types.

7. Illustration and Validation

Continuing from Section 3.1, we identified 15 change scenarios that cover changes in the content and changes in the ontologies. Based on the frequency of the change, their cascaded impacts, the operations involved and the number of ontologies affected, we ranked the change scenarios. For the purpose of this discussion, we use one of the change operations and present the impact analysis process.

For the purpose of the evaluation, the ontology layer contains 80 classes, 8 data properties, 10 object properties and more than 500 axioms. The annotation layer contains more than 1000 annotations and the content layer contains around 1000 HTML and XML documents.

The selected change operation is *Delete Class (Activity)*. When we delete *Activity*, we followed the proposed strategies in Section 4.2 and identified that the requested change operation can be implemented in five different strategies each yielding different set of atomic change operations. The five strategies are Attach-All, Attach-TBox, Cascade-All, Cascade-TBox and No-Action. We do not provide the detailed list of the change operations, however the number of change operations generated to implement the change in each strategy and the impacts of the change operations in each strategies are presented in Figure 8.

Figure 8 provides all the structural and semantic impacts of the change operations in each strategy. The change impact analysis method provides the detailed impacts of the change operations of each strategy. As a comparison, from the structural impact point of view, there are more axioms deleted in the two Cascade strategies (69) and the No-Action strategy deletes 18 axioms. The least is the Attach-All strategy with 5 axioms deleted followed by the Attach-TBox strategy which deletes 6 axioms. Another significant information is that both Cascade strategies delete 29 classes as a result of cascading. In all other strategies, only the intended class is removed. These and other analysis values show that in terms of structural impact, the Attach-All strategy is preferable because it has the minimum number of structural impacts. In light of the semantic impacts, the two Attach strategies yield the same result except on the instances.

In the Attach-All strategy there is one instance generalized and two instances less described. In Attach-TBox strategy there is no instance generalized, however 3 instances less described. This is due to the fact that the Attach-All strategy attaches all the instances to the parent class and results in generalization. In the Attach-TBox strategy, the instance is left without any further action, thus becomes less described as one of its previous parent is removed. In the case Cascade-All and Cascade-TBox strategies, there are more classes less described, instances less described and object properties less described. In general, the analysis in terms of semantic impact shows that the Attach-TBox strategy yields the minimum semantic impact and Cascade-TBox yields the worst semantic impact.

In this situation, making a comparison and a selection between the different impacts is difficult and time consuming. Moreover, the ontology engineer needs to compare between the different impacts. For example, class less described and orphan class may not be considered equally. An optimal strategy selection is the next stage to compare the different strategies in a further detail.

Once the change impact analysis is implemented, we further analyse these impacts following the optimal strategy selection. For the purpose of the case study we use two scenarios. The first scenario assigns equal weight to each of the four criteria (severity, statement type, operation type and performance) and the second assigns different weights. However, the system allows the ontology engineer to assign different weights to each criterion based on the nature of the OCMS at hand.

Scenario 1. This scenario assigns equal weight for each criterion. This means each criterion is given a weight of 0.25. The results of the four criteria are presented in Figure 9. The first graph shows the number of addition, deletion and total number of change operations. The Attach-All and Attach-TBox strategies have both addition and deletion operations. The Attach-All strategy has 13 additions and 19 deletions and the Attach-TBox strategy has 12 additions and 19 deletions. The No-Action strategy gives the least number of change operations which is only 19 deletions. But when we look at the Cascade strategies, both of them have large number of deletions. The results in terms of statement types show that the No-Action strategy yields the minimum number of changes on ABox and TBox statements. The largest number of TBox and ABox changes is observed on Cascade-All strategy because this strategy deletes all the dependent classes and instances. Attach-All and Attach-TBox strategies yield a similar result on TBox changes. The Attach-All strategy has one more ABox statement than the Attach-TBox strategy.

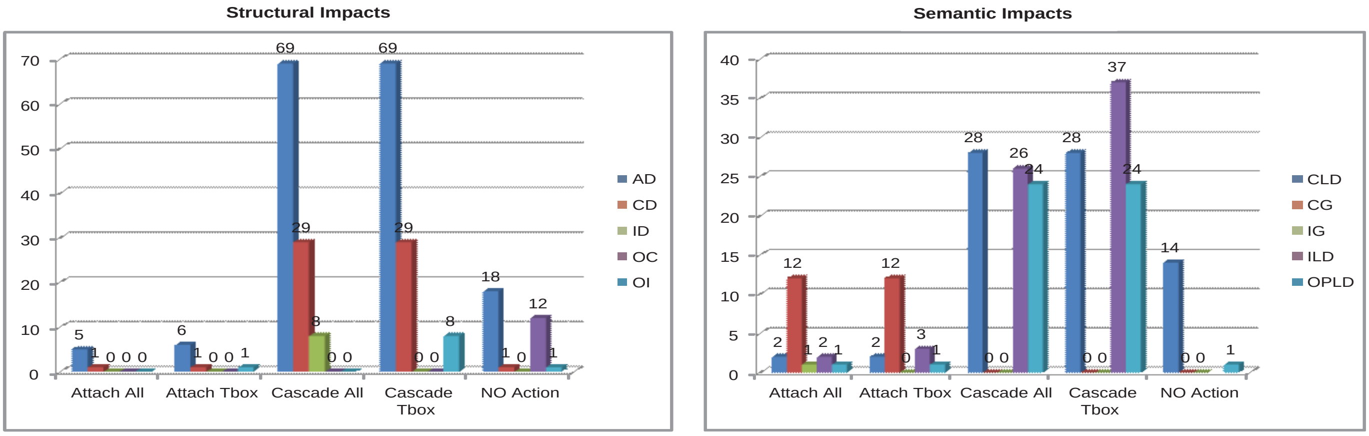

Figure 8: Structural and semantic impacts

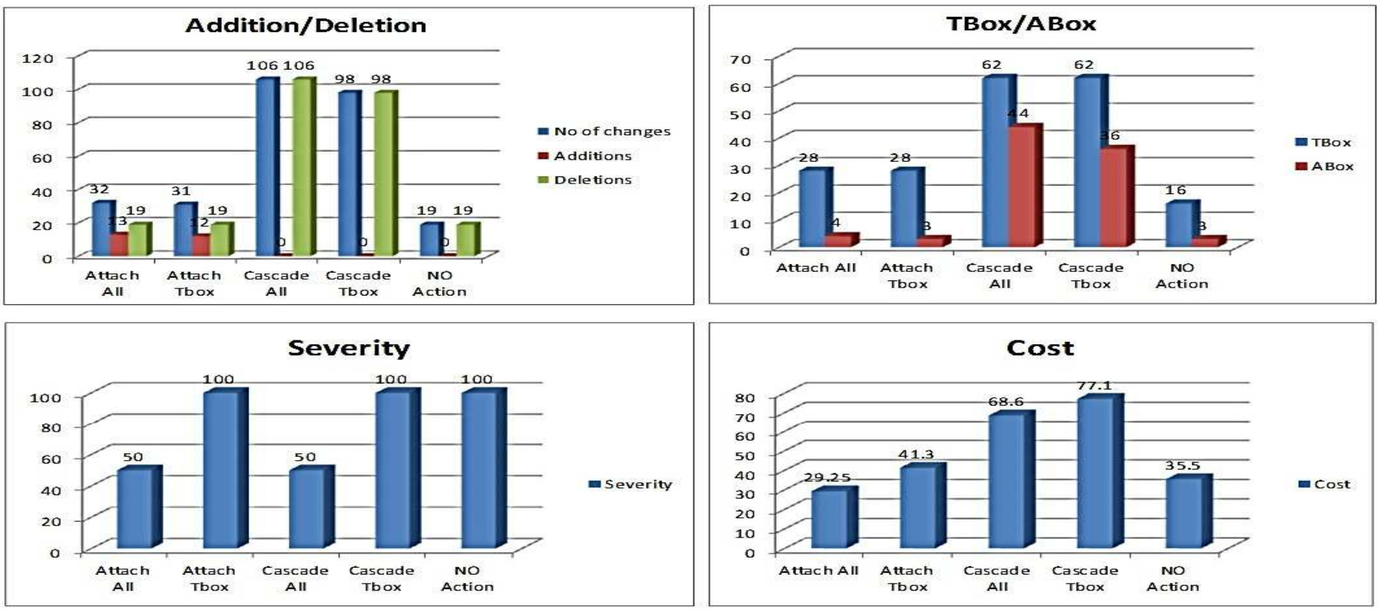

Figure 9: Cost of Evolution: scenario 1

If we look at the severity of the semantic and structural impacts based on the severity specification set to this case study, the Attach-All and the Cascade-All impacts show less severe impacts and all the rest indicate the presence of severe impacts that violate the constraints set by the ontology engineer. Finally, the cost of evolution takes all the four criteria into consideration by assigning similar weight to each criterion. The analysis result shows that the Attach-All strategy yields the minimum cost of evolution which makes it the best strategy to implement the change. The No-Action strategy ranks second and the Attach-TBox strategy ranks third. The Cascade-All strategy ranks fourth and the Cascade-TBox strategy ranks last.

Scenario 2. This scenario assigns different weights for each criterion for the purpose of the experiment. The weights for $S, ST, OT and P$ are 0.5, 0.3, 0.1, 0.1 respectively.

In this scenario, the weight assigned to the four criteria is different. Here we give more weight for the severity (0.5) and the statement types (0.3). This means, we want to avoid strategies that cause severe impacts than strategies that affect the statement types. The operation type and the performance are given equal weight. The results in Figure 10 shows the best strategy with minimum cost is Attach-All strategy (35.8). The second best strategy is No-Action strategy which yields 56.25 and the third one is Cascade-All strategy (58.4).

Scenario 1 and scenario 2 selected the Attach-All strategy as the best strategy and the No-Action strategy as the second best strategy. However, if we consider the rank, we can see that the third and the fourth strategies are swapped. Due to the increase of the weight of the severity criteria, the cost of evolution increases by some factor in all strategies used in scenario 2.

The change impact analysis method allows the ontology engineer to set his/her own weight depending on the requirement at hand. It further allows the user to evaluate the strategies using the individual criterion separately. One important feature of the approach is, it provides customizable change impact analysis by allowing users to assign severity values for the impacts, and weights for the criteria. By doing so, they can give necessary emphasis to impacts that are severe in a given OCMS and rank the criteria according to their importance in a given OCMS.

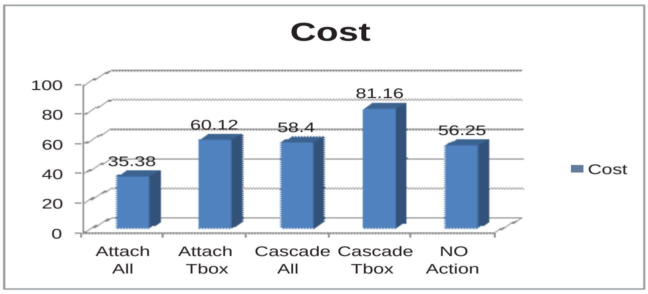

Figure 10: Cost of evolution: scenario 2

8. Evaluation

Our evaluation focuses on answering the following questions.

- Does the change impact analysis identify the impacts of the change operations accurately?
- Does the proposed method identify the optimal strategy accurately?
- Does the proposed method improve evolution of an OCMS?
- Does the change impact analysis analyse the impacts adequately?

For the purpose of the evaluation, we include two ontology evolution experts, one general ontology user and one novice user. All the users were given sufficient time to evaluate the change impact analysis tool and compare the results with protege and NeON ontology editors. For the purpose of evaluating the precision of the change impact analysis process, we used 10 frequent change operations taken from three different OCMS (Table 8). The users implement the changes using a prototype developed for change impact analysis and optimal strategy selection. An accurate change impact analysis should satisfy the following criteria. First, it should identify the impacts that actually occur in the OCMS and second it should identify the impacted entity correctly. When both criteria are satisfied, we consider the process accurate.

The First OCMS is the one presented in the case study. The second and the third OCMS cover a university administration domain and a database systems domain. These change operations represent frequent scenarios and are used to evolve the ontology using the applicable strategies. We implemented the change operations and measure the precision of the change impact analysis method. The result shows that the change impact analysis validated with 100% accuracy in all of the cases.

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.

Table 8: Identification of the first three optimal strategies

Change Operation	Optimal Strategy Identified?		
	1^{st}	2^{nd}	3^{rd}
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*)	√	√	-

8.1. Precision of Optimal Strategy Selection

We evaluate whether the proposed method achieves its objective by evaluating optimal strategy selection. The evaluation mainly focuses on checking whether the system identifies the optimal solution. We select the optimal solution according to the weight of the criteria set by the user of the OCMS. For the 10 change scenarios used above, 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 8 shows the evaluation result. The (√) 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 implemented changes.

8.2. Usefulness of the System

To evaluate the usefulness of the change impact analysis (CIA) framework and the change impact optimization (CIO) framework at the end of each scenario, we distributed a questionnaire to the participants. The questionnaire aims at answering whether the change impact analysis is useful and suitable for selecting optimal strategy to evolve an OCMS (Table 9). The separate presentation of the impacts of individual and composite change operations is vital to understand the impacts of the changes and to conduct what-if analysis.

Table 9: Users feedback on the optimal strategy selection

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
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 on the usability of the system.

- 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 result shows that the users strongly agree or agree on the adequacy of the solution. In both cases the respondents agree on the occurrence of the impacts. The result from the questionnaire shows that the change impact analysis method identifies the impacts and the affected entities. This helps the user to 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, the responses from the users are encouraging. 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.

Some of these users; however, focused on the presentation of the impacts (user interface issue) which is not the primary concern of the evaluation. 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. A comparative presentation of the alternative strategies in a single view is an important aspect. In general, the system provides us with an encouraging result in relation to selecting an optimal strategy.

8.3. Comparison of the System with other Systems

For further validation of the results, we compare our approach with the evolution approach used in Protege and NeON ontology editors. The comparison results are summarized in Table 10. The results show that our approach is complementary to existing tools and is capable of achieving transparency, reversibility and optimal implementation.

9. Conclusion

Changes in ontologies and OCMS cause different impacts on entities and dependent systems. It is difficult to manually identify the structural and semantic impacts of these changes. In this paper we present a change impact analysis method which analyses the structural and semantic impacts of atomic and composite change operations. The analysis includes identifying unsatisfiable entities and invalid instances. We further presented a novel approach to estimate the cost of evolving an OCMS using four crucial criteria. The optimal strategy selection is flexible and customizable to fit the specific requirements of the ontology engineers and corresponds to the OWL profiles.

The proposed approach benefits users by enabling them to view the actual impacts of change operations and the causes of the impacts. It further enables them to compare implementation strategies in terms of impact. It allows transparent evolution by providing the impacts of changes as individual and/or composite operations. This approach assists the user to conduct what-if analysis before permanently implement a requested change operation. The system is able to compare and select an optimal strategy that ensures consistent evolution whenever there is an alternative implementation strategy. It uses criteria such as severity of impacts, number of change operations, statement types and operation types.

The integration of our change impact analysis tool with existing ontology editors and development of a plug-in and the integration of additional criteria for optimal strategy selection such as impacts on inferred axioms is the future direction of this research.

Acknowledgment. This material is based upon works supported by the Science Foundation Ireland under Grant No. 07/CE/I1142 as part of the Centre for Next Generation Localisation (www.cngl.ie) at Dublin City University (DCU).

References

[1] H.-C. Chu, M.-Y. Chen, Y.-M. Chen, A semantic-based approach to content abstraction and annotation for content management, Expert Syst. Appl. 36 (2009) 2360–2376.
[2] T. R. Gruber, A translation approach to portable ontology specifications, Knowledge Acquisition 5 (1993) 199–220.
[3] N. F. Noy, M. Klein, Ontology evolution: Not the same as schema evolution., Knowledge and Information Systems. 6 (2004) 328–440.
[4] V. Benjamins, J. Contreras, O. Corcho, A. Gomez-perez, Six challenges for the semantic web., Cristani, M(ED): KR2002 Workshop on the Semantic Web, Toulouse, France. (2002).
[5] L. Stojanovic, Methods and tools for ontology evolution., Ph.D. thesis, University of Karlsruhe, 2004.
[6] V. Gruhn, C. Pahl, M. Wever, 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, Springer-Verlag, London, UK, 1995, pp. 270–281.
[7] G. Konstantinidis, G. Flouris, G. Antoniou, V. Christophides, A formal approach for rdf/s ontology evolution, in: Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence, IOS Press, Amsterdam, The Netherlands, The Netherlands, 2008, pp. 70–74.
[8] L. Qin, V. Atluri, Evaluating the validity of data instances against ontology evolution over the semantic web., Information and Software Technology. 51 (2009) 83–97.
[9] A. Khattak, Z. Pervez, S. Lee, Y.-K. Lee, After effects of ontology evolution, in: Future Information Technology (FutureTech), 2010 5th International Conference on, pp. 1 –6.
[10] M. Klein, Change Management for Distributed Ontologies, Ph.D. thesis, Vrije Universiteit Amsterdam, 2004.
[11] P. Plessers, O. De Troyer, Ontology change detection using a version log, in: Proceedings of the 4th international conference on The Semantic Web, ISWC'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 578–592.
[12] Y. Abgaz, M. Javed, C. Pahl, 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, 2010.
[13] Y. Abgaz, M. Javed, C. Pahl, 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, 2011.
[14] Y. Abgaz, M. Javed, C. Pahl, 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*, 2012, pp. 3–12.

Table 10: Comparison of impacts of different implementation strategies

Criteria	Protege	Neon	Ours
Structural Impact	does not show details of structural impact	shows structural changes	shows changes, impacts, impacted entities and gives explanation
Semantic Impact	does not show semantic impact before implementation	does not show semantic impact before implementation	shows impacts, impacted entities and gives explanation
Transparency	user don't know which entities are affected before the change	structural impacts are transparent but not semantic impacts	users are able to see detailed impacts of atomic or composite changes (how and why)
Implementation strategy	Delete target entity and delete entity and its reference	allow composition by adding or removing atomic changes	allows Attach-All, Cascade, No-Action for TBox and ABox whenever applicable
Optimal strategy suggestion	Not available	Not Available	compares and shows the optimal strategy
Reversibility	provides undo and redo	provides undo and redo	provides undo and redo

[15] Y. Abgaz, M. Javed, C. Pahl, 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.

[16] Y. Abgaz, Change Impact Analysis for Evolving Ontology-based Content Management., Ph.D. thesis, School of Computing, Dublin City University, 2013.

[17] L. Stojanovic, N. Stojanovic, S. Handschuh, 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.

[18] N. F. Noy, A. Chugh, W. Liu, M. A. Musen, A framework for ontology evolution in collaborative environments, in: 5th International Semantic Web Conference, Springer-LNCS, 2006, pp. 544–558.

[19] P. Plessers, O. De Troyer, S. Casteleyn, Understanding ontology evolution: A change detection approach, Web Semant. 5 (2007) 39–49.

[20] G. Flouris, D. Plexousakis, Handling ontology change: Survey and proposal for a future research direction, Artificial Intelligence (2005) 1–55.

[21] G. Flouris, D. Plexousakis, G. Antoniou, A classification of ontology change., Poster Proceedings of the 3rd Italian Semantic Web Workshop, Semantic Web Applications and Perspectives(SWAP-2006) (2006).

[22] N. F. Noy, M. A. Musen, Promptdiff: A fixed-point algorithm for comparing ontology versions., in: AAAI/IAAI'2002, pp. 744–750.

[23] T. Tudorache, N. F. Noy, S. Tu, M. A. Musen, Supporting collaborative ontology development in protege, in: Proceedings of the 7th International Conference on The Semantic Web, ISWC '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 17–32.

[24] T. Redmond, M. Smith, N. Drummond, T. Tudorache, Managing change: An ontology version control system, in: In OWL: Experiences and Directions, 5th Intl. Workshop, OWLED 2008.

[25] T. Redmond, N. Noy, Computing the changes between ontologies, in: Workshop on Knowledge Evolution and Ontology Dynamics, ISWC 2011.

[26] E. Jiménez Ruiz, B. C. Grau, I. Horrocks, R. Berlanga, Supporting concurrent ontology development: Framework, algorithms and tool, Data Knowl. Eng. 70 (2011) 146–164.

[27] E. J. Ruiz, B. C. Grau, I. Horrocks, R. Berlanga, Building ontologies collaboratively using contentcvs, in: Proceedings of the 22nd International Workshop on Description Logics (DL 2009).

[28] M. Hartung, A. Gross, E. Rahm, CODEX: Exploration of semantic changes between ontology versions, Bioinformatics 26 (2012) 895–896.

[29] L. Zhang, S. Xia, Y. Zhou, Z. Xia, User defined ontology change and its optimization, in: Control and Decision Conference, 2008. CCDC 2008. Chinese, pp. 3586 –3590.

[30] V. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, F. Ciravegna, 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 (2006) 14–28.

[31] C. Pahl, M. Javed, Y. Abgaz, Utilising ontology-based modelling for learning content management, in: Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010, AACE, Toronto, Canada, 2010, pp. 1274–1279.

[32] M. Şah, V. Wade, Automatic metadata extraction from multilingual enterprise content, in: Proceedings of the 19th ACM international conference on Information and knowledge management, CIKM '10, ACM, New York, NY, USA, 2010, pp. 1665–1668.

[33] P. Ceravolo, E. Damiani, M. Viviani, Bottom-up extraction and trust-based refinement of ontology metadata, IEEE Transactions on Knowledge and Data Engineering 19 (2007) 149 –163.

[34] L. Baresi, R. Heckel, Tutorial introduction to graph transformation: A software engineering perspective, in: Proceedings of the First International Conference on Graph Transformation, ICGT '02, Springer-Verlag, London, UK, 2002, pp. 402–429.

[35] R. Heckel, Graph transformation in a nutshell, Electronic Notes in Theoretical Computer Science 148 (2006) 187–198.

[36] J. Trinkunas, O. Vasilecas, A graph oriented model for ontology transformation into conceptual data model, Technology 36 (2007) 126–132.

[37] V. Bönström, A. Hinze, H. Schweppe, Storing rdf as a graph, in: Proceedings of the First Conference on Latin American Web Congress, LA-WEB '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 27–36.

[38] H. Zhang, Y.-F. Li, H. B. K. Tan, Measuring design complexity of semantic web ontologies, J. Syst. Softw. 83 (2010) 803–814.

[39] R. Palma, P. Haase, O. Corcho, A. Gmez-prez, Change representation for owl 2 ontologies, in: Proceedings of the sixth international workshop on OWL: Experiences and Directions (OWLED).

[40] M. Lee, A. J. Offutt, R. T. Alexander, 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, IEEE Computer Society, Washington, DC, USA, 2000, pp. 61–70.

[41] R. S. Arnold, Software Change Impact Analysis, IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[42] L. Cox, D. Harry, D. Skipper, H. S. Delugach, Dependency analysis using conceptual graphs, in: Proceedings of the 9th International Conference on Conceptual Structures, ICCS 2001, Springer, 2001.

[43] R. Volz, D. Oberle, S. Staab, B. Motik, 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, 2003.

[44] H. Knublauch, R. Fergerson, N. Noy, M. Musen, 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*, Springer Berlin / Heidelberg, 2004, pp. 229–243.

[45] M. O. Hassan, L. Deruelle, H. Basson, 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.

[46] G. Castagna, Covariance and contravariance: conflict without a cause, ACM Transactions on Programming Languages and Systems 17 (1995) 431–447.

[47] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, New York, NY, USA, 2003.

[48] R. S. Goncalves, B. Parsia, U. Sattler, Analysing the evolution of the nci thesaurus, in: Proceedings of the 2011 24th International Symposium on Computer-Based Medical Systems, CBMS '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 1–6.

[49] J. L. Johnson, Probability and Statistics for Computer Science, John Wiley & Sons, Hoboken, 2011.

[50] K. S. Trivedi, Probability and Statistics with Reliability, Queuing and Computer Science Applications, John Wiley and Sons Ltd., Chichester, UK, 2nd edition edition, 2002.

[51] J. Sacks, W. Welch, T. Mitchell, H. Wynn, Design and Analysis of Computer Experiments, Statistical science 4 (1989) 409–423.