CHAPTER 1

Ontology Evolution and the Referencing of Resources in Semantic Web Context

Delia ROGOZAN and Gilbert PAQUETTE
LICEF Research Center, TELUQ, Québec, CANADA

In Dicheva, D., Mizoguchi, R., & Greer, J. (Eds.). (2009). *Ontologies and Semantic Web for e-Learning* (p. 5-23). Amsterdam, The Netherlands : IOS Press.

Abstract. Because ontologies evolve over time, their evolution needs to be managed. Therefore, in this paper, we propose a framework composed of two main systems: ChangeHistoryBuilder, which tracks and manages the history of ontology changes, and SemanticAnnotationModifier, which provides a support to maintain the integrity of the ontology-based referencing of resources after the ontology evolution. Both systems are based on a formal specification of types of possible changes in OWL-DL ontologies. In concrete terms, this specification is an ontology of ontology changes.

Keywords. Ontology Evolution, Ontology of Ontology Changes, Tracking Changes, Managing the Ontology-based Referencing of Resources

Introduction

Evolution is a fundamental requirement for useful ontologies. Since ontologies are knowledge theories of a precise domain, they need to evolve because the domain has changed or because problems in the original domain conceptualization have to be resolved [1]. Moreover, in open and dynamic environments such as the Semantic Web, the ontologies need to evolve because domain knowledge evolves continually [2] or because ontology-oriented software-agents must respond to changes in users' needs [3]. Consequently, ontology evolution is an essential part of research in ontology engineering and in application of ontologies in Semantic Web environments.

This chapter explores some important issues of ontology evolution. Three research questions structure this chapter: (1) what is ontology evolution and which are the types of possible changes in OWL ontologies; (2) how can we manage the evolution history by logging changes brought to ontologies; (3) what are the effects of changes on the ontology-based referencing of resources and how can we resolve them?

1. Ontology Evolution and Ontology Changes

1.1. Definition of the Ontology Evolution Notion

Actual research is far from defining the ontology evolution notion in a consensual way. Thus, for the authors of [4, 5], the ontology evolution signifies the process of applying changes to a unique ontology, while the authors of [6, 7] consider it more as the building and the management of multiple ontology versions. Both interpretations are pertinent in the distributed and dynamic context of the Semantic Web.

Consequently, we consider the *ontology evolution* as the timely modification of an ontology by application of changes to an ontology version ($\mathbf{V_N}$) in order to obtain a new ontology version ($\mathbf{V_{N+1}}$), while preserving the ontology consistency and roles. The *ontology role* refers to the service provided by the ontology and to its usage. For example, in the Semantic Web context, the ontology is used to assure the semantic referencing so that resources can be found by the knowledge they contain [8, 9]. The *ontology consistency* designates the state where all structural and axiomatic constraints of the ontology model are respected.

An *ontology change* is a modification brought to ontology during the evolution from a version V_N to a new version V_{N+1}. Changes can be elementary or complex. An *elementary change* is a simple and non-composite change (*i.e.* addition or deletion of ontology elements). A *complex change* is a collection of elementary changes, which form together a logical entity whose signification is unique and clearly defined (*cf.* Table 1).

Table 1. Examples of complex changes

Complex change	**Collection of elementary changes**
`MergeClasses` (C_1 … C_N) into class C	- `DeleteClass` (C_1), …, `DeleteClass` (C_N) - `AddClass` (C)
`SplitClass` C into classes (C_1 … C_N)	- `DeleteClass` (C) - `AddClass` (C_1), …, `AddClass` (C_N)
`ModifySuperClass` of C, from class A to B	- `DeleteSuperClass` (A) from (C) - `AddSuperClass` (B) to (C)
`MoveDisjointClass` (C), from class A to B	- `DeleteDisjointClass` (C) from class A - `AddDisjointClass` (C) to class B

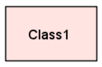
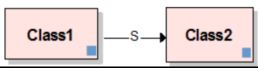
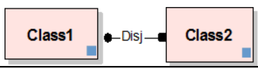
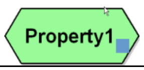
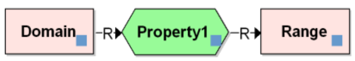
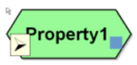
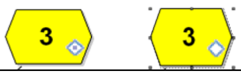
Application of changes can induce inconsistencies in other parts of the ontology [10]. For example, merging two classes will cause subclasses and properties to be inconsistent. Resolving that problem can be treated as a request for new additional changes, e.g. subclasses and properties can be either deleted, or attached to some other classes. Thus, a *primary change* is not a consequence of any change, while an *additional* one is caused by another change (named parent-change).

1.2. An Ontology of Ontology Changes

The notion of “change” is central to ontology evolution. Indeed, to describe evolution it is necessary to specify formally all types of changes that can be applied to ontologies. Regarding the change specification, actual research proposes only taxonomies of elementary changes, although complex changes have a richer semantic [11].

In this section, we propose an ontology of changes that can be applied in OWL-DL ontologies. This ontology expands the taxonomies described in [12, 13] by adding a typology of complex changes, as well as a number of properties and axioms. We have built this ontology with the MOWL[1] graphical editor developed by a LICEF team. The following table presents some of the basic graphical symbols used by MOWL to represent ontologies.

Table 2. An example of some of the graphical symbols used by MOWL to represent ontologies

MOWL Symbol	Description	MOWL Symbol	Description
	Class		Subclass axiom
			Disjoint classes axiom
	Property		Property with domain and range
	Functional Property		Exact/ Minimal Cardinality

1.2.1. Classification of Changes in Change Ontology

We present here an extract of our ontology; more details can be found in [14]. The Change Ontology consists of two principal hierarchies. The `ChangeObject` hierarchy specifies the ontology objects that can be changed, *i.e.* elements used to build OWL ontologies, such as classes, properties or axioms. The `ChangeOperation` specifies the types of changes in OWL-DL ontologies. It consists of two taxonomies, one of elementary changes and one of complex changes, both of them being described further.

1.2.1.1. Operations of Elementary Changes

The taxonomy of elementary changes contains the generic changes `Add_Change` and `Delete_Change`. The conceptual structure of these generic changes is similar. For that reason, in Figure 1 we illustrate only the classification of additions.

The changes that add elements are classified according to their application object: `Add_To_Ontology`, `Add_To_Class`, `Add_To_Property`. From the ‘ontology’ point of view, there are two main changes: `Add_Class` and `Add_Property`. From the ‘class’ point of view there are multiple changes: additions of logical axioms (*i.e.* `intersectionOf`, `complementOf`, `unionOf`), additions of class axioms (*i.e.* `superClass`, `equivalentClass`, `disjointWith`) or even additions of property restrictions that characterize classes. From the ‘property’ point of view, the main changes operate on the property domain and range as well as on the property axioms (*i.e.* `superProperty`, `equivalentProperty`, `inverseProperty`).

1.2.1.2. Operations of Complex Changes

The taxonomy of complex changes contains the main types of complex changes, which are those that merge, split, modify or move elements of ontologies `(Merge_Change`,

[1] The MOWL is a tool for editing OWL ontologies and for exporting them to XML files compliant with the OWL-DL language (http://www.cogigraph.com/Produits/OWLDLOntologyEditors/tabid/1100/language/en-US/Default.aspx).

`Split_Change`, `Modify_Change`, `Move_Change`). Other types of complex changes are those that add, delete or modify sub-hierarchies of OWL elements (`Subtree_Change`). Given the number of concepts represented by this taxonomy (more than 50), we present in Figure 2 only the classification of the `Modify_Change` type.

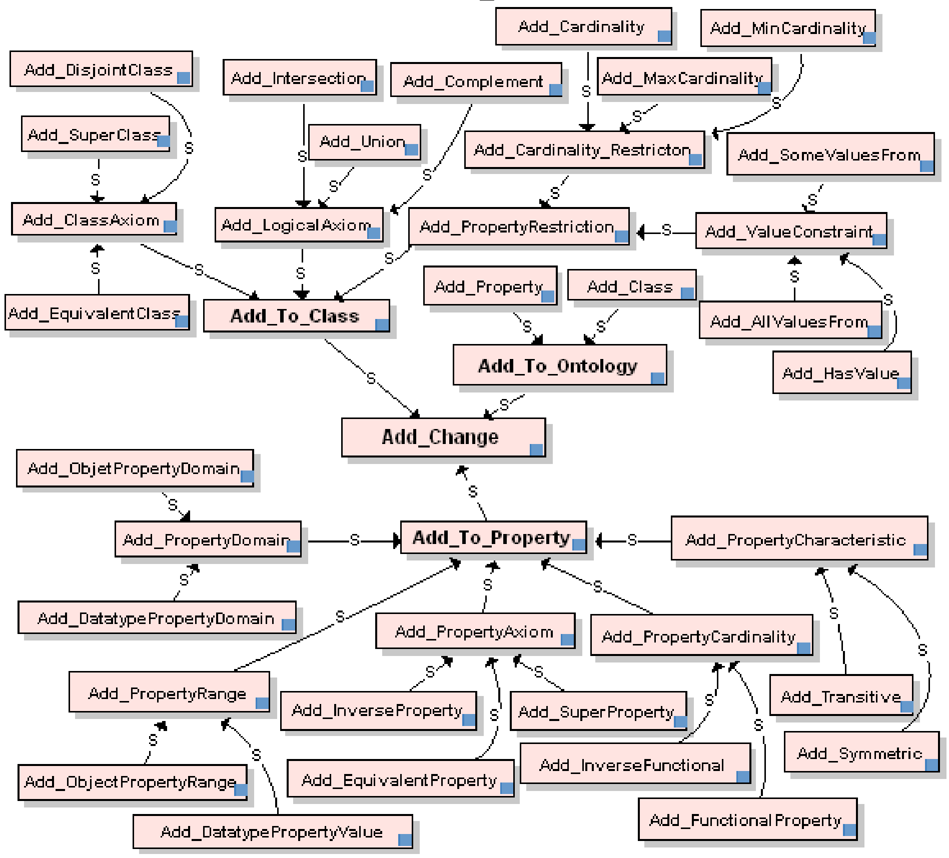


Figure 1. Classification of elementary changes that add elements to OWL-DL ontologies

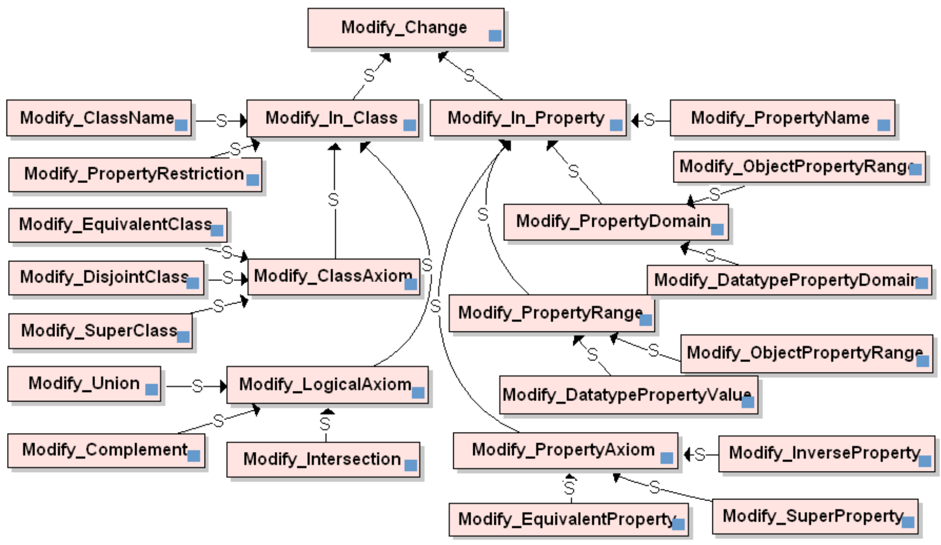


Figure 2. Classification of complex changes that modify elements in OWL-DL ontologies

1.2.2. Change Characterization in Change Ontology

To allow a richer characterization of changes, we also defined some properties.

1.2.2.1. Properties of Changes

Figure 3 introduces the general properties of change operations. The `appliedOn` property connects the change operations to ontology objects. The properties `haveSource` and `haveTarget` describe the source and the target of change operations. Both properties have as domain a class of type `ChangeOperation` and as range a class of type `ChangeObject` or a value `rdfs:Datatype`.

Other properties are introduced, as well: `haveChangeNumber`, which specifies a reference number that indicates the application order of a change and `haveParentChangeNumber`, which declares the reference number of the parent-change.

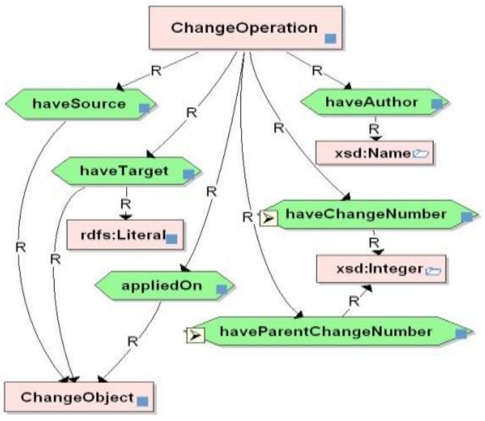


Figure 3. Properties of change operations

1.2.2.2. Characterization of Changes by means of Property Restrictions

Restrictions on these general properties may be associated to each change operation in order to characterize it formally. Figure 4 shows a part of the characterization of `Add_Change`, the same method being followed for all other changes.

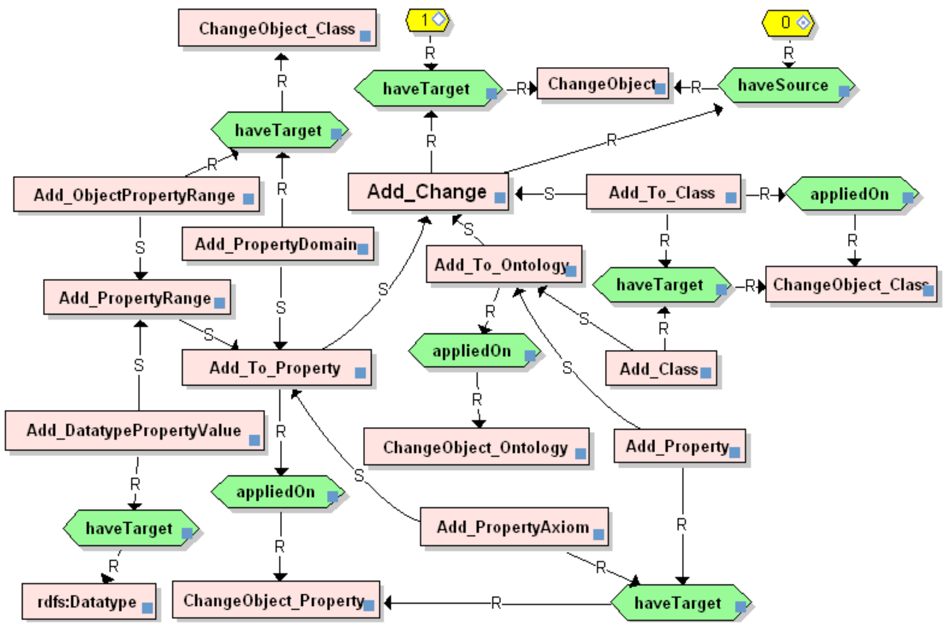


Figure 4. Part of the characterization of addition changes

Thus, any operation of type `Add_Change` is characterized by two general restrictions: an exact cardinality of 0 for the `haveSource` property, which declares that there is no element as source of an addition change; a minimal cardinality of 1 for the `haveTarget` property, which declares that the target of any addition change comprises at least one element. The application object is defined by adding restrictions on `appliedOn` property. These restrictions characterize changes that add elements to the ontology structure (`Add_To_Ontology`), to a class definition (`Add_To_Class`) or to a property definition (`Add_To_Property`).

We described in this section an ontology of ontology changes that extends previous classifications. It also adds clear definitions of change operations by means of properties. We can now use this formal theory to support the development of tools for managing ontology changes. We address this objective in the following sections where we propose two interlinked systems for managing (1) the history of ontology changes and (2) the ontology-based referencing of resources after the ontology evolution.

2. Managing the History of Ontology Changes with ChangeHistoryBuilder (CHB)

Although the management of ontology changes is one of the key issues in successful applications of evolving ontologies, methods and tools to support it are almost missing [15]. As we underlined in [14], very little research concerning tools for keeping track of ontology changes has been carried out. However, these tools are important to consider for ontology-based referencing of resources since changes affect the way that resources should be handled and interpreted by means of new ontology versions.

There are two major approaches for tracking and managing ontology changes. The first one logs changes during ontology evolution [4, 13]. Even if this approach facilitates later retrieval of all performed changes, it presents an important problem: the log-files are stored independently from ontology versions and a tool-oriented language formalizes them. Consequently, these log-files are more difficult to identify, access and interpret by Semantic Web agents. For that reason, the second approach relies only on a comparison between ontology versions to identify changes [12, 16]. However, it presents a problem as well. It can identify only some elementary changes and therefore, it cannot provide complete information about evolution processes[2].

The *ChangeHistoryBuilder (CHB)* system overcomes these two problems: it combines the fact of having access to a log that captures the entire semantic of ontology evolution with the fact of identifying changes starting only from ontology versions. It also can deal with complex changes, in addition to elementary ones.

To track and manage the history of ontology changes, the CHB system supports a four-step process, as illustrated in Figure 5.

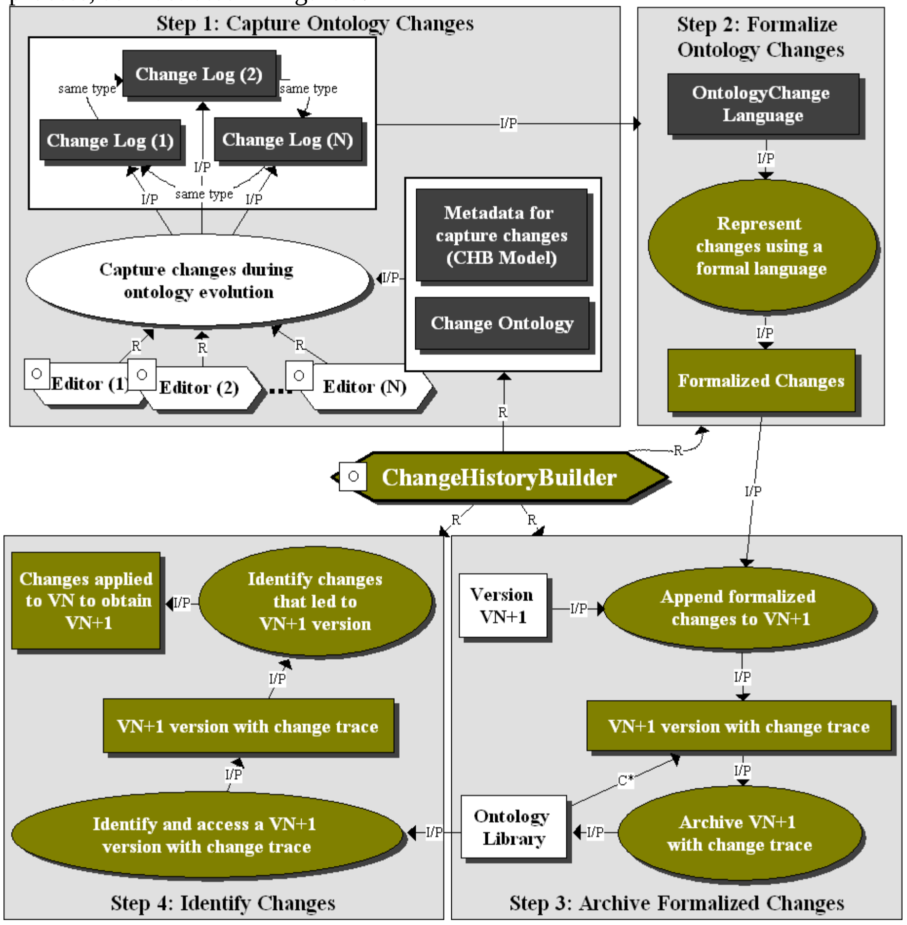

Step 1: Capture Ontology Changes
Change Log (2)
same type
same type
Change Log (1)
Change Log (N)
I/P
I/P
same type
I/P
Capture changes during ontology evolution
I/P
Metadata for capture changes (CHB Model)
Change Ontology
R
R
R
Editor (1)
Editor (2)
Editor (N)
I/P
Step 2: Formalize Ontology Changes
OntologyChange Language
I/P
Represent changes using a formal language
I/P
Formalized Changes
R
R
ChangeHistoryBuilder
I/P
R
R
Changes applied to VN to obtain VN+1
I/P
Identify changes that led to VN+1 version
I/P
VN+1 version with change trace
I/P
Identify and access a VN+1 version with change trace
I/P
Step 4: Identify Changes
Version VN+1
I/P
Append formalized changes to VN+1
I/P
VN+1 version with change trace
C*
I/P
Ontology Library
I/P
Archive VN+1 with change trace
Step 3: Archive Formalized Changes

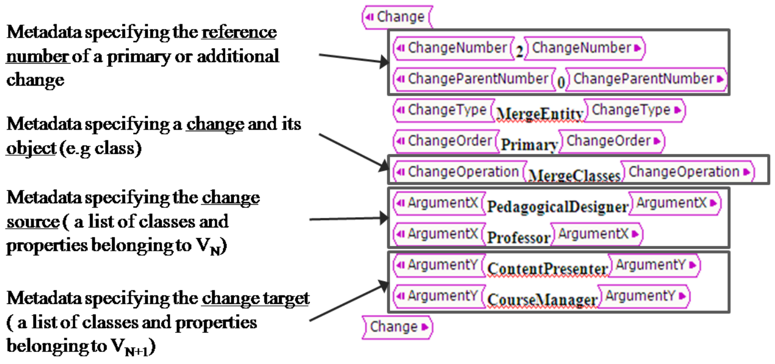
Figure 5. The fourth-step functioning of the CHB system

Legend of the graphical formalism [17] : procedure (oval shape), input/output resource of a procedure (rectangular shape) and actor that carries out the underling procedure (hexagonal shape); link composition (C), specialization (S), precedence (P), input-output (I/P) and regulation (R).

[2] Knowing that two classes were deleted from V_N does not tell us that these classes were merged in V_{N+1}.

2.1. *Capturing Changes during Ontology Evolution (Step 1)*

The first step aims at logging in a log-file all changes applied during the evolution from V_N to V_{N+1}. To resolve the interpretation problems of log-files generated by different editors, the CHB provides ontology editors with a uniform and common model for logging changes. The CHB model is a set of metadata that aggregates in a common structure all changes [18]. Based on the change ontology, these metadata allow ontology editors to capture specific information about elementary and complex changes, in addition to general information regarding the ontology version. These ontology editors can use the CHB model as a plug-in and thereby generate log-files presenting a normalized and rich description of applied changes. A log-file example of this sort is presented in Figure 6.


Metadata specifying the reference number of a primary or additional change
Metadata specifying a change and its object (e.g class)
Metadata specifying the change source (a list of classes and properties belonging to V_N)
Metadata specifying the change target (a list of classes and properties belonging to V_{N+1})

```
<Change>
  <ChangeNumber>2</ChangeNumber>
  <ChangeParentNumber>0</ChangeParentNumber>
  <ChangeType>MergeEntity</ChangeType>
  <ChangeOrder>Primary</ChangeOrder>
  <ChangeOperation>MergeClasses</ChangeOperation>
  <ArgumentX>PedagogicalDesigner</ArgumentX>
  <ArgumentX>Professor</ArgumentX>
  <ArgumentY>ContentPresenter</ArgumentY>
  <ArgumentY>CourseManager</ArgumentY>
</Change>
```

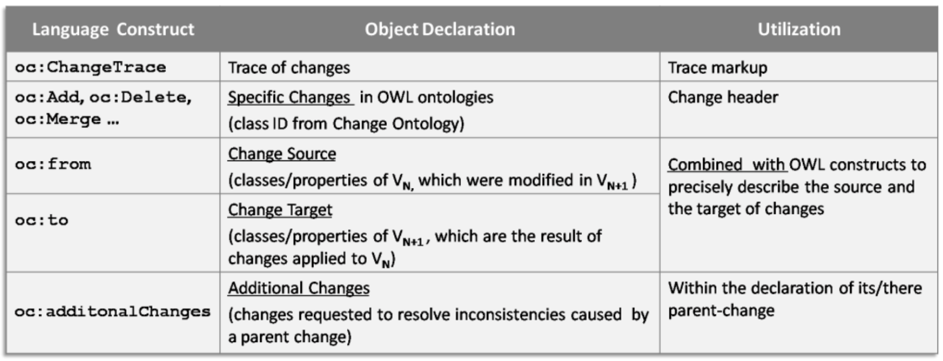
Figure 6. Example of a log-file based on the CHB model

Despite the figure above, the CHB model is not a linear one. It is organized so that every primary change is represented under a tree-shape that is formed by additional changes. Moreover, this change tree is generated in a flexible way according to evolution strategies applied by ontologists during evolution.

2.2. *Formalizing Changes using OC+OWL Language (Step 2)*

The second step of the logging process supported by the CHB system is the formalization of changes that were captured during the previous step. For this purpose, we developed a formalization language, named *OntologyChange* (OC), which is based on a minimal number of constructs, labelled oc. When combined to those of OWL [19], these constructs formally describe all types of changes in OWL-DL ontologies. Table 3 shows a concise summary of OC language constructs and Figure 7 illustrates how CHB uses these constructs to formalize changes. Consequently, all semantic web agents or software components, which are able to manipulate the OC+OWL language, can also interpret and reason with the trace of formalized changes that were logged using the CHB model.

Table 3. OC language constructs to formalize changes

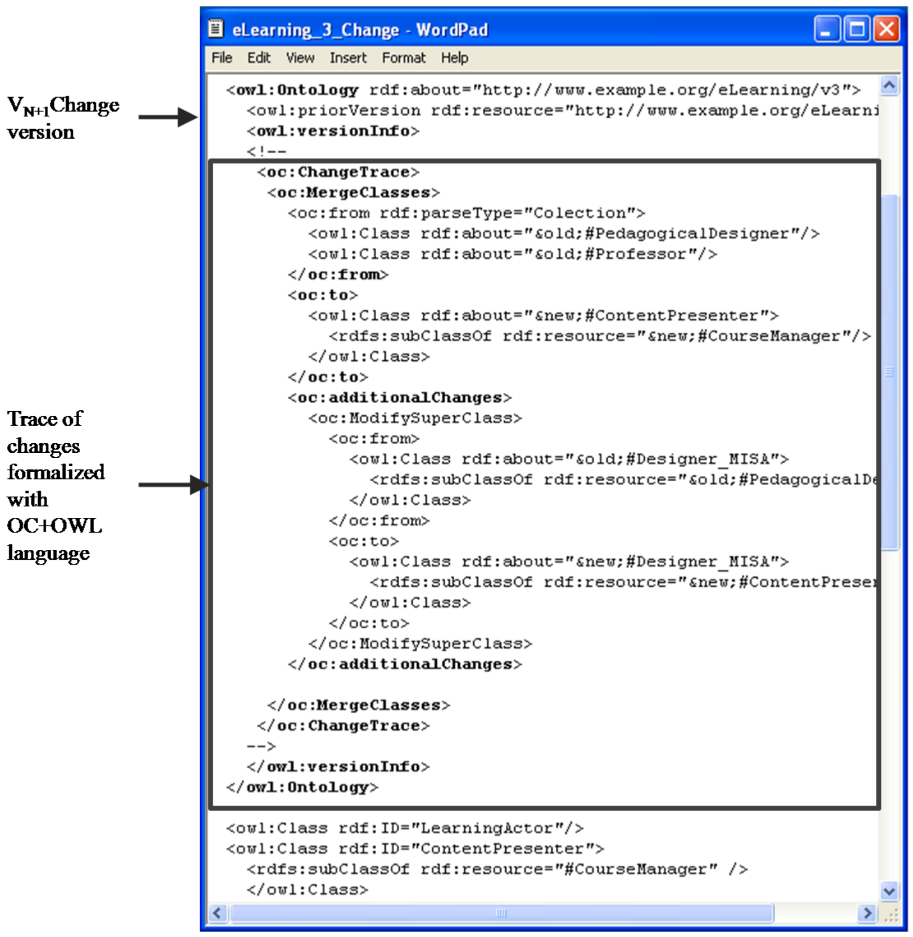


Language Construct	Object Declaration	Utilization
`oc:ChangeTrace`	Trace of changes	Trace markup
`oc:Add`, `oc:Delete`, `oc:Merge` …	Specific Changes in OWL ontologies (class ID from Change Ontology)	Change header
`oc:from`	Change Source (classes/properties of V_N, which were modified in V_{N+1})	Combined with OWL constructs to precisely describe the source and the target of changes
`oc:to`	Change Target (classes/properties of V_{N+1}, which are the result of changes applied to V_N)	
`oc:additonalChanges`	Additional Changes (changes requested to resolve inconsistencies caused by a parent change)	Within the declaration of its/there parent-change

2.3. Archiving Formalized Changes in the New Ontology Version (Step 3)

The third step consists in the archiving of previous formalized changes. The solution proposed by the CHB system is to append to the new ontology version the trace of changes formalized with OC+OWL language. The expression *V_{N+1}Change* denotes thus this new ontology version with an integrated trace of changes. In this way, V_{N+1}Change contains in addition to the underlying domain conceptualization, all information about the evolution from V_N to V_{N+1}. Figure 7 presents an example of a V_{N+1}Change version.

In order to preserve the interpretation of V_{N+1}Change through all OWL compliant tools, the formalized changes are declared inside the `owl:versionInfo` statement. According to OWL language, this statement gives information about ontology versions without contributing to the logical meaning of the ontology. The resulting V_{N+1}Change version thus conforms to the OWL language, while offering information about all applied changes.


eLearning_3_Change - WordPad
File Edit View Insert Format Help
V_{N+1}Change version
Trace of changes formalized with OC+OWL language

```
<owl:Ontology rdf:about="http://www.example.org/eLearning/v3">
  <owl:priorVersion rdf:resource="http://www.example.org/eLearni
  <owl:versionInfo>
  <!--
   <oc:ChangeTrace>
    <oc:MergeClasses>
      <oc:from rdf:parseType="Colection">
        <owl:Class rdf:about="&old;#PedagogicalDesigner"/>
        <owl:Class rdf:about="&old;#Professor"/>
      </oc:from>
      <oc:to>
        <owl:Class rdf:about="&new;#ContentPresenter">
          <rdfs:subClassOf rdf:resource="&new;#CourseManager"/>
        </owl:Class>
      </oc:to>
      <oc:additionalChanges>
        <oc:ModifySuperClass>
          <oc:from>
            <owl:Class rdf:about="&old;#Designer_MISA">
              <rdfs:subClassOf rdf:resource="&old;#PedagogicalDe
            </owl:Class>
          </oc:from>
          <oc:to>
            <owl:Class rdf:about="&new;#Designer_MISA">
              <rdfs:subClassOf rdf:resource="&new;#ContentPrese
            </owl:Class>
          </oc:to>
        </oc:ModifySuperClass>
      </oc:additionalChanges>

    </oc:MergeClasses>
   </oc:ChangeTrace>
  -->
  </owl:versionInfo>
</owl:Ontology>

<owl:Class rdf:ID="LearningActor"/>
<owl:Class rdf:ID="ContentPresenter">
  <rdfs:subClassOf rdf:resource="#CourseManager" />
  </owl:Class>
```

Figure 7. An example of a V_{N+1}Change ontology version

2.4. Identifying Changes Starting from the New Ontology Version (Step 4)

The fourth step concerns the interpretation of changes after the ontology evolution. The CHB system is able to identify all applied elementary and complex changes, together with their primary-additional relationship, by simply reading the OC+OWL trace contained in the V_{N+1}Change ontology version. Furthermore, all software agents able to interpret OC+OWL language can also identify changes starting only from V_{N+1}Change.

3. Managing Semantic Referencing with SemanticAnnotationModifier (SAM)

Ontology evolution can give rise to side effects on the resource referencing and can thus hamper one of the most important features of the Semantic Web: the ontology-based referencing of resources that formally describes the resources content.

Consider the example of the ontology evolution from V_N to V_{N+1} and a resource R_1, which is referenced by the class `PedagogicalDesigner` belonging to V_N. During evolution, this class is merged with another class and consequently, it no longer exists in the new ontology version. This makes resource R_1 no more accessible for requests of type "Give me a resource which is a `PedagogicalDesigner`": the access to R_1 is broken via V_{N+1}. Consider furthermore a resource R_2 that is referenced by two classes `Tutor` and `Researcher`. If a disjunction axiom is added between these two classes, then the interpretation of R_2 becomes inconsistent via the new ontology version.

However, despite the necessity of managing the effects of ontology changes on the resource referencing, little research tackled this issue. For example, in [2] it was demonstrated that the add changes do not affect the access to referenced data, while changes that delete entities hamper it. Or, the authors of [20] analyzed the effects of elementary changes on the class hierarchy. The authors of [21] analyzed and proved that modifications made on an ontology whose concepts are used to generate metadata may disrupt the metadata semantic. In [22] was proposed the CREAM annotation model together with some recommendations regarding the modification of resource referencing, yet without proposing any concrete solution to that purpose. The authors of [23] presented a rule-based approach to detect and correct inconsistencies of ontology-based semantic annotations. Finally, let us underline that, even if a wide range of referencing tools exists nowadays, none of them is able to support an evolving ontology-based referencing of resources.

In this context, the second system that we propose in this chapter is as much innovatory as fundamental. The *SemanticAnnotationModifier (SAM)* system provides a support for managing the ontology-based referencing of resources after the ontology evolution. In order to present SAM, we start by explaining the notion of semantic referencing on which the system is based. Then, we discuss the operation model of SAM and we explain it through examples.

3.1. Semantic Referencing of Resources by means of UKIs

Semantic referencing denotes the description of resources content by means of formal semantic descriptors. These descriptors, named *semantic references*, are generally knowledge, *i.e.* classes according to the OWL terminology, belonging to different ontologies.

To specify the semantic references, we use the URI general syntax. A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies all kind of objects, whether they are physical (e.g. images, documents, services, actors) or abstract (e.g. concepts in an ontology). It consists of a hierarchical sequence of common components: `scheme`, `authority`, `path` and `fragment` [24].

In addition, to assert that semantic references identify solely ontology concepts, we introduce the terms *Uniform Knowledge Identifier (UKI)* and we define it as a URI with two restrictions: the first three components must identify a unique version of an ontology and the last component must identify a unique class inside this ontology version. Thus, as illustrated in Figure 8, an UKI is composed of two principal elements: the URI of the ontology version and the name of a class within this version.

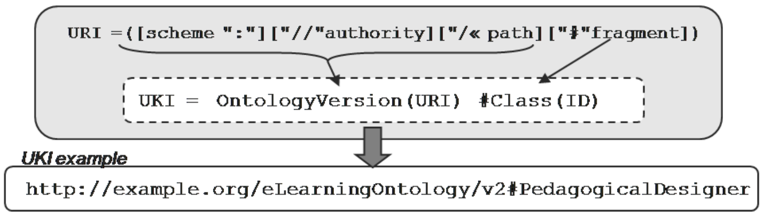

URI = ([scheme ":"]["//"authority]["/« path]["#"fragment])
UKI = OntologyVersion(URI) #Class(ID)
UKI example
http://example.org/eLearningOntology/v2#PedagogicalDesigner

Figure 8. An UKI that specifies the reference `PedagogicalDesigner` within the second version of `eLearningOntology`

In conclusion, the semantic referencing consists of one or several semantic references associated to resources to describe their content formally, each reference being specified by means of a UKI (*cf.* Figure 9).

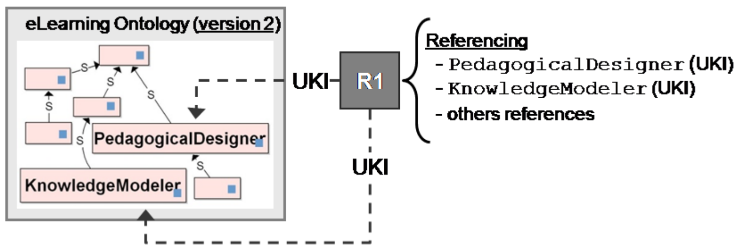


Figure 9. The semantic referencing of a resource

3.2. Operation Model of the SAM System

We present the operation model of the SAM system in Figure 10. This model underlines the two main services that SAM offers to users. The first one analyses changes applied to V_N to obtain V_{N+1}. The purpose here is to inform users about changes that hinder the access to referenced resources or that modify their interpretation. The second service modifies the semantic referencing (*e.g.* UKIs) that is affected by ontology changes. The purpose here is to allow access to all resources via the new ontology version as well as a consistent interpretation of them. Both services are based on data provided by the CHB system, consisting of complete and semantically rich information about elementary and complex changes together with the causality relation existing among them.

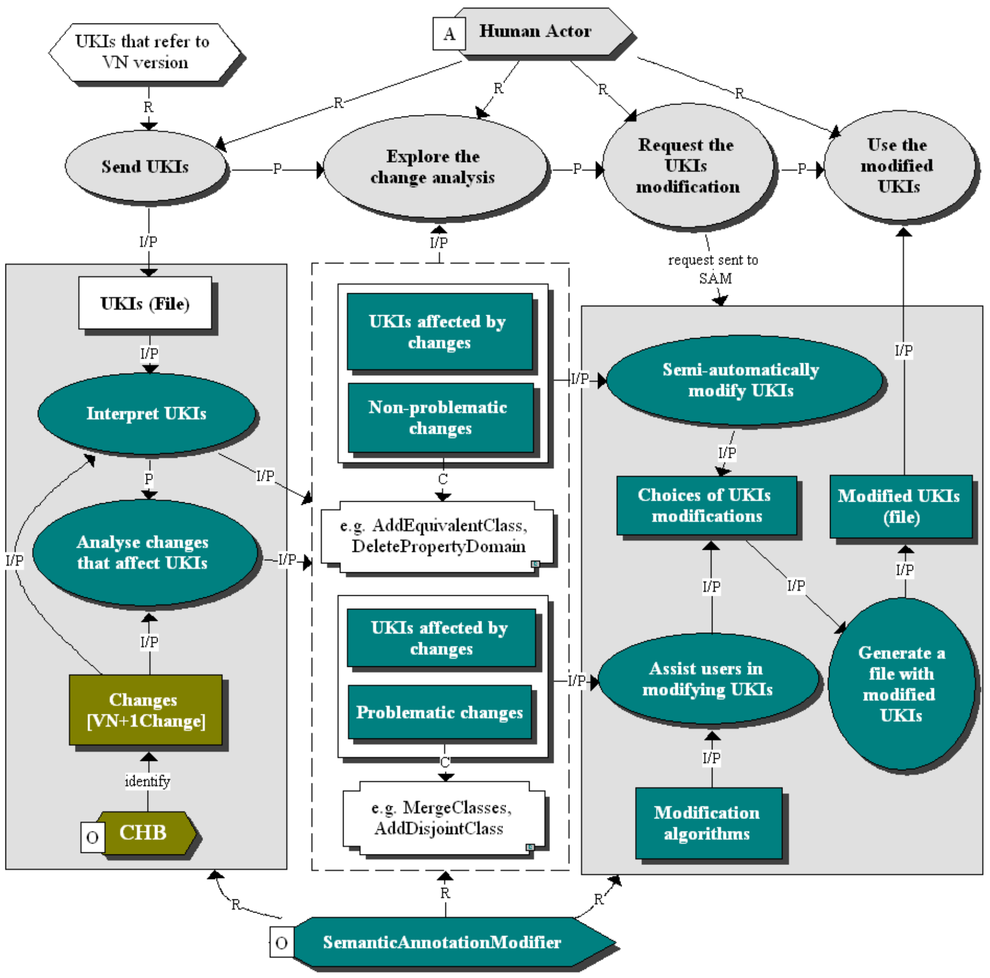

UKIs that refer to VN version
A
Human Actor
R
Send UKIs
Explore the change analysis
Request the UKIs modification
Use the modified UKIs
P
I/P
request sent to SAM
UKIs (File)
Interpret UKIs
Analyse changes that affect UKIs
Changes [VN+1Change]
identify
O
CHB
UKIs affected by changes
Non-problematic changes
C
e.g. AddEquivalentClass, DeletePropertyDomain
UKIs affected by changes
Problematic changes
e.g. MergeClasses, AddDisjointClass
Semi-automatically modify UKIs
Choices of UKIs modifications
Modified UKIs (file)
Assist users in modifying UKIs
Generate a file with modified UKIs
Modification algorithms
O
SemanticAnnotationModifier

Figure 10. The operation model of the SAM system (see legend of Figure 5)

3.3. Exemplifying the Operation Model of SAM

In this section, we exemplify the operation model of SAM. We start by illustrating how SAM analyses the change effects on resource referencing. Next, we present how SAM assists users in modifying this resource referencing.

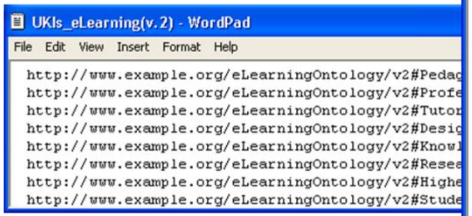
3.3.1. SAM Analysis the Change Effects on Resource Referencing

3.3.1.1. Users send UKIs to SAM in order to analyse them

Let us consider a user who wants to verify if the semantic referencing of a resource collection is affected by the evolution from an ontology version V_N to a new version V_{N+1}. In that purpose, he sends to SAM a file containing the UKIs (*i.e.* references) associated to these resources. For this first prototype of SAM, we imposed some constraints on the file format: the UKIs file must stem from the same owner; it must be organized as a list; all UKIs must refer to the same ontology version.

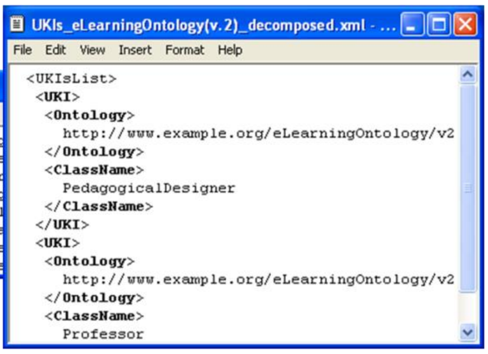
3.3.1.2. SAM interprets UKIs

To interpret the UKIs file, SAM decomposes every UKI in order to identify the URI of the V_N ontology version together with the name of the class used as reference (*cf.* Figure 11).



UKIs_eLearning(v.2) - WordPad

```
http://www.example.org/eLearningOntology/v2#Pedag
http://www.example.org/eLearningOntology/v2#Profe
http://www.example.org/eLearningOntology/v2#Tutor
http://www.example.org/eLearningOntology/v2#Desig
http://www.example.org/eLearningOntology/v2#Knowl
http://www.example.org/eLearningOntology/v2#Resea
http://www.example.org/eLearningOntology/v2#Highe
http://www.example.org/eLearningOntology/v2#Stude
```



UKIs_eLearningOntology(v.2)_decomposed.xml - ...

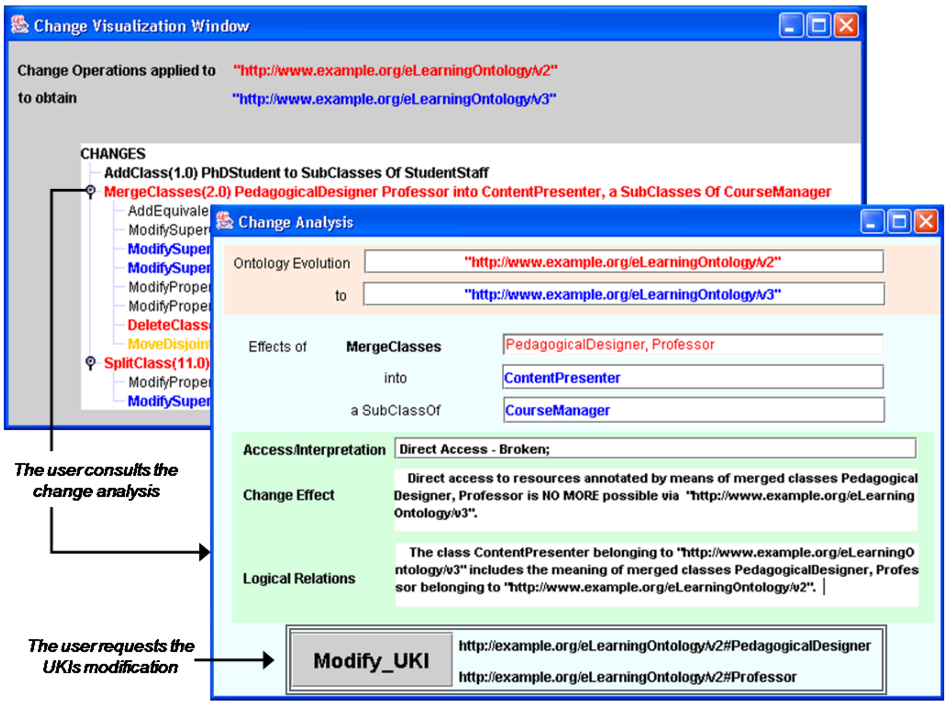
```
<UKIsList>
 <UKI>
  <Ontology>
    http://www.example.org/eLearningOntology/v2
  </Ontology>
  <ClassName>
    PedagogicalDesigner
  </ClassName>
 </UKI>
 <UKI>
  <Ontology>
    http://www.example.org/eLearningOntology/v2
  </Ontology>
  <ClassName>
    Professor
```

Figure 11 (a). User UKIs file **Figure 11 (b).** Decomposed UKIs

Then, it asks CHB for the V_{N+1}Change and extracts all ontology changes that were applied to V_N to obtain V_{N+1}. Because SAM can interpret the OC+OWL language, it can also 'understand' the trace of changes appended to V_{N+1}Change. Finally, SAM links UKIs to changes by matching each class name specified by UKIs to its corresponding pair in the change trace.

3.3.1.3. SAM analyses change effects on UKIs and the user request UKIs modification

Based on UKIs interpretation, SAM presents to user an analysis of changes (*cf.* Figure 12).


Change Visualization Window
Change Analysis
The user consults the change analysis
The user requests the UKIs modification
Modify_UKI

Change Operations applied to "http://www.example.org/eLearningOntology/v2"
to obtain "http://www.example.org/eLearningOntology/v3"

CHANGES
- AddClass(1.0) PhDStudent to SubClasses Of StudentStaff
- MergeClasses(2.0) PedagogicalDesigner Professor into ContentPresenter, a SubClasses Of CourseManager
  - AddEquivale
  - ModifySuper
  - ModifySuper
  - ModifySuper
  - ModifyPrope
  - ModifyPrope
  - DeleteClass
  - MoveDisjoin
- SplitClass(11.0)
  - ModifyPrope
  - ModifySuper

Ontology Evolution "http://www.example.org/eLearningOntology/v2"
to "http://www.example.org/eLearningOntology/v3"

Effects of MergeClasses PedagogicalDesigner, Professor
into ContentPresenter
a SubClassOf CourseManager

Access/Interpretation: Direct Access - Broken;

Change Effect: Direct access to resources annotated by means of merged classes Pedagogical Designer, Professor is NO MORE possible via "http://www.example.org/eLearning Ontology/v3".

Logical Relations: The class ContentPresenter belonging to "http://www.example.org/eLearningOntology/v3" includes the meaning of merged classes PedagogicalDesigner, Professor belonging to "http://www.example.org/eLearningOntology/v2".

http://example.org/eLearningOntology/v2#PedagogicalDesigner
http://example.org/eLearningOntology/v2#Professor

Figure 12. Change visualization and change analysis with SAM

Firstly, SAM highlights: (1) changes that break the access to resources, in red; (2) changes that give rise to an inconsistent interpretation of resources, in yellow; (3) changes that modify the interpretation of resources (e.g. by modifying the class-parent of the class used as semantic reference), in blue.

Secondly, SAM provides the user with an analysis of effects for each underlined change. This analysis[3] consists of three panels. The two first ones deal with the effect of changes on the access to referenced resources or on the consistency of their interpretation. The third one indicates the relation exiting between a class belonging to V_N and the same or other class belonging to V_{N+1}, according to criteria as identity, equivalence, inclusion, generalization, specialization or conceptually different. This last panel is particularly useful for understanding how the meaning of a class used as reference was modified during the ontology evolution.

Starting from the change analysis, the user has the possibility to request the modification of resource referencing (*i.e.* UKIs) that is affected by ontology changes.

3.3.2. SAM Modifies the Resource Referencing

3.3.2.1. SAM modifies the resource referencing affected by non problematic changes

This modification concerns the UKIs affected by changes that do not cause either a loss of access to resources, or an inconsistent interpretation of them. Changes of this type are `AddEquivalentClass` or `ModifySuperClass`, for example.

Thus, to allow access to resources via the new ontology version, SAM modifies only the URI of the ontology version inside UKIs (*cf.* example bellow). The user has to validate it, even though this modification can be automatically processed.

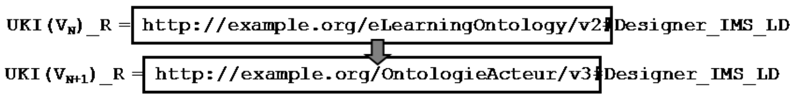

UKI(V_N)_R = http://example.org/eLearningOntology/v2#Designer_IMS_LD
UKI(V_N+1)_R = http://example.org/OntologieActeur/v3#Designer_IMS_LD

Figure 13. Modification of the UKI referring to `Designer_IMS_LD` (*cf.* evolution example from Figure 15)

3.3.2.2. SAM identifies several solutions for the modification of the resource referencing affected by problematic changes

This situation concerns especially the UKIs affected by changes that hamper the access to resources via the new ontology version (e.g. `MergeClasses`, `DeleteClass`, `SplitClass`). In this case, most of classes used as references in UKIs are no more available in V_{N+1}. To give access to resources, SAM should then modify, besides the URI of the ontology version, the class name in each affected UKI (*cf.* example bellow).

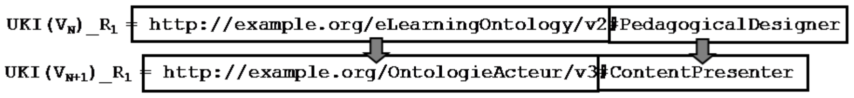

UKI(V_N)_R1 = http://example.org/eLearningOntology/v2#PedagogicalDesigner
UKI(V_N+1)_R1 = http://example.org/OntologieActeur/v3#ContentPresenter

Figure 14. Modification of the UKI referring to `PedagogicalDesigner` (*cf.* evolution example from Figure 15)

However, this modification cannot be automatically processed because several solutions are possible. To detect them, SAM exploits two identification algorithms that we developed in [14]. Since these algorithms are based on the information provided by the V_{N+1}Change, they are able to deal with all problematic changes. Consider, for example, the

[3] As we are focusing on the general functioning of SAM, we are not going to discuss the change analysis in this chapter. Details can however be found in [14].

change `MergeClasses` illustrated in Figure 15. SAM is able to detect several classes that can be pertinent for the UKIs modification:

– *Classes semantically "closed",* such as the `ContentPresenter` because it includes the meaning of `PedagogicalDesigner`.
– *First-level subclasses* of classes the name of which must be modified in UKIs. Regarding our example, these subclasses are `Designer_MISA` and `Designer_IMS_LD`. They were transferred to another class in V_{N+1}, after the removal of their parent-class.
– Classes to which first-level subclasses were transferred, *i.e.* `CourseManager`.

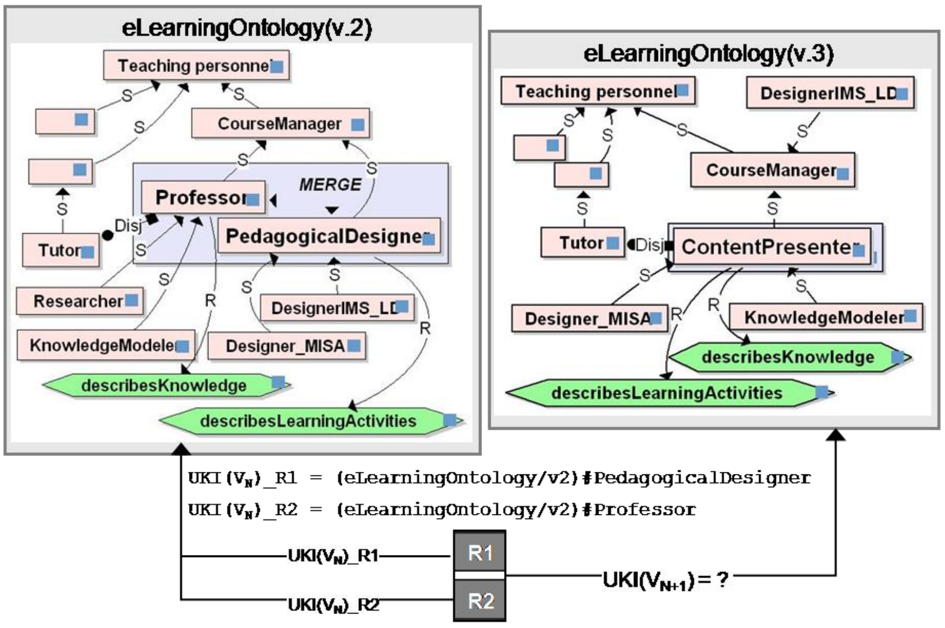
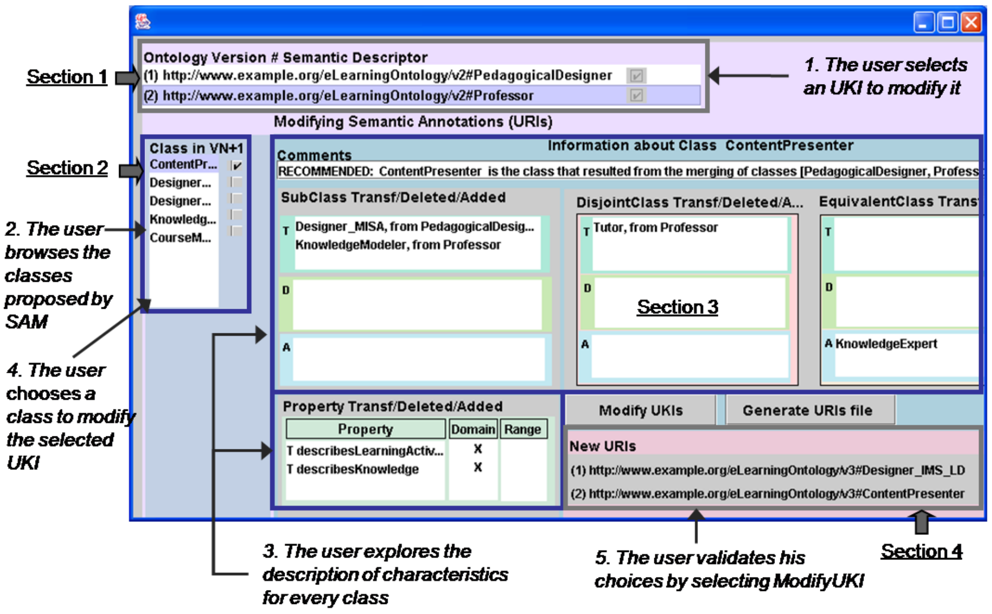


Figure 15. Identification of pertinent classes for the modification of UKIs affected by problematic changes: the `MergeClasses` example.

3.3.2.3. SAM assists users in modifying UKIs affected by problematic changes

Choosing among the solutions identified by SAM is the user privilege; only the user may decide which solution is more appropriated to his context. However, SAM can guide him during the modification process. The Figure 16 presents the interaction between the SAM system and a user who wants to modify UKIs affected by problematic changes (the example of `MergeClasses` is considered).

As shown in this figure, the SAM interface consists of four principal sections. *Section 1* indicates the UKIs affected by the change whose analysis was previously explored by the user. *Section 2* presents the classes identified as being pertinent for the modification of indicated UKIs. The classes are enumerated in a decreasing order, according to their pertinence degree. *Section 3* consists of comments and specific characteristics of classes listed in Section 2. For each class, the "Comments" panel describes the reason why a class was considered by SAM as pertinent. The other panels indicate the subclasses, axioms and properties that were deleted from, transferred or added to the selected class. Finally, *Section 4* presents the modified UKIs.


Section 1
Section 2
Section 3
Section 4
1. The user selects an UKI to modify it
2. The user browses the classes proposed by SAM
3. The user explores the description of characteristics for every class
4. The user chooses a class to modify the selected UKI
5. The user validates his choices by selecting ModifyUKI
Modifying Semantic Annotations (URIs)
Class in VN+1
Information about Class ContentPresenter
Comments
SubClass Transf/Deleted/Added
DisjointClass Transf/Deleted/A...
EquivalentClass Trans
Property Transf/Deleted/Added
Property
Domain
Range
Modify UKIs
Generate URIs file
New URIs

Ontology Version # Semantic Descriptor
(1) http://www.example.org/eLearningOntology/v2#PedagogicalDesigner
(2) http://www.example.org/eLearningOntology/v2#Professor

ContentPr... ✓
Designer...
Designer...
Knowledg...
CourseM...

RECOMMENDED: ContentPresenter is the class that resulted from the merging of classes [PedagogicalDesigner, Profess

T Designer_MISA, from PedagogicalDesig...
KnowledgeModeler, from Professor
D
A

T Tutor, from Professor
D
A

T
D
A KnowledgeExpert

Property	Domain	Range
T describesLearningActiv...	X	
T describesKnowledge	X	

(1) http://www.example.org/eLearningOntology/v3#Designer_IMS_LD
(2) http://www.example.org/eLearningOntology/v3#ContentPresenter

Figure 16. Interaction between SAM and users during the modification of UKIs

3.3.2.4. SAM generates the file of modified UKIs
Once the user validated the modification of all UKIs, SAM generates a file containing these modified UKIs and sends it to the user.

4. Evaluation and Deployment of CHB and SAM in eLearning Contexts

4.1. Evaluation of CHB and SAM Systems

Regarding the systems evaluation, we carried out a technical validation of CHB and SAM with ontologies of small and average size, a diversified set of changes and the UKIs files that respect the specified constraints.

We also conducted a qualitative evaluation of CHB and SAM systems according to the utility criterion, *i.e.* a criterion allowing to identify, for a given context, the interest and the relevance degree of systems features [25]. In our case, the general target context was the Semantic Web. The specific context was that of eLearning systems based on ontologies and on the semantic referencing of resources. We used several techniques while undergoing the evaluation of systems, i.e. thinking aloud method, qualitative questionnaires, interviews and a focus-group. Six participants were then selected. They all have knowledge of OWL ontologies as well as experience in the eLearning fields. The evaluation took place in the LORIT[4] laboratory for observation, test and experimentation of instructional technologies.

In order to draw some valid meaning from qualitative data that we collected during systems evaluation, we based our analysis on the method proposed in [26]. Some outcomes of this data analysis are illustrated in Table 4. Other results may be found in [14].

[4] LORIT (http://www.licef.teluq.uquebec.ca/lorit/eng/Index.htm) stands for Research Laboratory-Observatory in Tele-learning Engineering.

Table 4. Qualitative evaluation of CHB and SAM systems: some outcomes

CHB	**SAM**
The utility was validated for both systems, especially for users that are responsible of the ontology management and of the ontology-based referencing of resources	
The evaluation participants demonstrated…	*The evaluation participants appreciated…*
A better understanding of the ontology evolution after the visualization of changes, especially in the case of complex changes	The fact that SAM allows users to control the referencing modification (for problematic changes)
	The relevance of multiples solutions proposed by SAM
The evaluation participants underlined orientations for future works, such as …	
Use the integrated trace of changes as a support to the collaborative modification of an ontology	Customize the assistance (or automatism) level of the modification of resource referencing with SAM
Connect a change viewer with an ontology viewer	Make available new means to define new references

4.2. Deployment of CHB and SAM Systems in eLearning Contexts

Once the evaluation of CHB and SAM completed, the next step is to deploy these systems in eLearning contexts. To that effect, we have selected the TELOS project that was designed and developed by a LICEF team within the LORNET research network [27].

The Technology Enhanced Learning Operating System (TELOS) aims to enable pedagogical technologists to develop, modify or use eLearning resources within a service-oriented framework. In TELOS, all types of 'content provider', e.g. multi-media document, learning object, learning design, knowledgeable person, are eLearning resources. All these resources are referenced using specific knowledge defined in domain ontologies. The goal here is to allow the search of relevant eLearning resources, the aggregation of resources according to their semantic description and the creation of consistent learning scenarios based on a semantic equilibrium among resources [28].

The ontology-based referencing layer is thus a foundational element of the TELOS framework. Considering that, CHB and SAM systems are necessary to manage the referencing of resources, given that domain ontologies are not fixed entities: at any moment, these ontologies may be modified by TELOS users according to their needs. Therefore, we illustrate in Table 5 the services that will be provided by CHB and SAM, once these modules are integrated into TELOS.

Table 5. Services provided by CHB and SAM in TELOS system

Services provided by CHB and SAM in TELOS system	**CHB**	**SAM**
Track changes during the modification of TELOS domain ontologies using the MOWL editor	√	
Help distant ontologists to see all changes made on a shared ontology	√	
Draw attention to the potential effect of a change in order to allow users to approve or to cancel it during ontology evolution /modification	√	√
Allow the exploration of change history after ontology evolution	√	
Automatically highlight the change effects on resource referencing, given that all resources are stored in TELOS repositories	√	√
Automatically update the resource referencing affected by non-problematic changes		√
Support users in modifying the resource referencing affected by problematic changes		√

5. Conclusion

We proposed in this article a framework for managing ontology changes and for resolving some of their problematic effects. This framework is composed of three major components: an ontology of changes, a system that tracks changes during ontology evolution (CHB) and a system that supports users in maintaining the semantic referencing of resources (SAM).

Building an ontology of ontology changes is an emergent preoccupation in our research domain. We therefore developed a representation of elementary and complex changes that can be applied to OWL-DL ontologies (more than 60 operations were identified and characterized by means of properties). Based on this representation, we conceived and built a first version of the CHB and SAM systems.

Concerning the CHB system, we underlined our three principal contributions. Firstly, based on the change ontology, we developed a model that allows ontology editors to capture elementary and complex changes in a uniform manner. Secondly, we proposed the OC language for change formalization. Using a minimal number of constructs, together with those of OWL, this language can represent formally all types of changes in OWL-DL ontologies. Thirdly, we offered a solution to problems of tools oriented log-files access and interpretation. This solution is to append the trace of formalized changes to the new ontology version in a manner that keep this version OWL compliant.

Regarding the SAM system, our principal contribution consisted in the exploration of new and essential ideas in the ontology-based referencing domain, *i.e.* an appropriate modification of resource referencing in order to allow access to all resources by means of the new ontology version. For this purpose, the SAM system offers solutions and guides the users during the process of referencing modification. It maps between the referencing of resources (*i.e.* UKIs set) and ontology changes in order to identify the affected UKIs. It analyses the change effects on the access and on the interpretation of resources. For UKIs affected by problematic changes, it identifies a set of concepts belonging to the new ontology versions, which can be pertinent for UKIs modification. Finally, it allows users to choose among these different solutions by giving them information about the appropriateness of each identified concept.

As we have completed the evaluation of prototypes for both CHB and SAM system, we currently aim to improve these two systems for making them able to treat all types of elementary and complex changes as well as different representation formats of semantic referencing. We also work on a project to integrate them in the TELOS system for eLearning and knowledge management.

References

[1] N. Noy and M. Klein, Ontology evolution: Not the same as schema evolution, *Knowledge and Information Systems* **5** (2003).

[2] J. Heflin and J. Hendler, Dynamic Ontology on the Web, 17th National Conference on artificial Intelligence (AAAI), 2000.

[3] L. Stojanovic, A. Maedche, N. Stojanovic, and R. Studer, Ontology Evolution as Reconfiguration- Design Problem Solving, Second International Conference on Knowledge Capture, 2003.

[4] A. Maedche, B. Motik, and L. Stojanovic, Managing Multiple and Distributed Ontologies in the Semantic Web, *VLDB Journal - Special Issue on Semantic Web*, **12** (2003), 286-302.

[5] L. Stojanovic and B. Motik, Ontology Evolution within Ontology Editors, Knowledge Acquisition, Modeling and Management (EKAW), Siguenza, Spain, 2002.

[6] M. Klein, Y. Ding, D. Fensel, and B. Omelayenko, Ontology management - Storing, aligning and maintaining ontologies, in *Towards the Semantic Web: Ontology-Driven Knowledge Management*, J. Davids, D. Fensel, and F. vanHarmele, Eds., Wiley, 2002, 47-69.
[7] N. Noy and M. Musen, Ontology Versioning as an Element of an Ontology-Management Framework, *IEEE Intelligent Systems* (2003).
[8] T. Berners-Lee, J. Hendler, and O. Lasilla, The Semantic Web, *Scientific American* **5** (2001), 34–43.
[9] J. Hendler, Agents and the Semantic Web, *IEEE Intelligent systems* **3/4** (2001), 30-37.
[10] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic, User-driven Ontology Evolution Management, 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02), Sigüenza, Spain, 2002.
[11] M. Klein and N. Noy, A component-based framework for the ontology evolution, Workshop on Ontologies and Distributed Systems, IJCAI-2003, Acapulco, Mexico, 2003.
[12] M. Klein, Change Management for Distributed Ontologies, Vrije Universiteit Amsterdam, 2004.
[13] L. Stojanovic, Method and tools for ontology evolution, University of Karlsruhe, Germany, 2004.
[14] D. Rogozan, Management of the ontology evolution: methods and tools for an evolving semantic referencing based on analysis of changes applied to ontology versions (in french), in *LICEF Center*, vol. PhD. Montréal: Université de Québec à Montréal (UQAM)/Télé-université (TELUQ), 2008.
[15] P. Haase and Y. Sure, State-of-the-Art on Ontology Evolution, Technical report, SEKT informal deliverable 3.1.1.b, Institute AIFB, University of Karlsruhe 2004.
[16] N. Noy, S. Kunnatur, M. Klein, and M. Musen, Tracking Changes During Ontology Evolution, 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan, 2004.
[17] G. Paquette, *Modélisation des connaissances et des compétences, pour concevoir et apprendre*: Presses de l'Université du Québec, 2002.
[18] D. Rogozan and G. Paquette, Managing Ontology Changes on the Semantic Web, IEEE/WIC/ACM International Conference on Web Intelligence (WI'05), Compiegne, France, 2005.
[19] W3C_WebOnt, OWL Web Ontology Language Guide and Reference, 2004.
[20] H. Stuckenschmidt and M. Klein, Integrity and change in modular ontologies., 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico, 2003.
[21] P. Ceravolo, A. Corallo, G. Elia, and A. Zilli, Managing Ontology Evolution Via Relational Constraints, Knowledge-Based Intelligent Information and Engineering Systems, 8th International Conference, KES, Wellington, New Zealand, 2004.
[22] S. Handschuh, Semantic Annotation of Resources in the Semantic Web, in *Semantic Web Services*, R. Studer, S. Grimm, and A. Abecker, Eds.: Springer Berlin Heidelberg, 2007, 135-155.
[23] H. Luong and R. Dieng-Kuntz, A rule-based approach for semantic annotation evolution, *Computational Intelligence* **23** (2007), 320-338.
[24] T. Berners-Lee, R. Fielding, and L. Masinter, Uniform Resource Identifier (URI): Generic Syntax, Network Working Group, 2005.
[25] J. Nielsen, *Usability engineering*: Boston, Academic Press, 1993.
[26] M. Miles and A. Huberman, *Qualitative Data Analysis (2nd edition)*. Thousand Oaks, CA: Sage Publications, 1994.
[27] G. Paquette, I. Rosca, S. Mihaila, and A. Masmoudi, Telos, a service-oriented framework to support learning and knowledge management, in *E-Learning Networked Environments and Architectures: a Knowledge Processing Perspective*, S. Pierre, Ed.: Springer-Verlag, 2007.
[28] G. Paquette and F. Magnan, Learning Resource Referencing, Search and Aggregation At the eLearning System Level, presented at IODE Workshop, ECTEL-07 Conference, Crete, September 18-21, 2007.