Management of Change in the Web Ontology Language

by

Figen Füsun Horozal

A thesis for conferral of a

Master of Science in Computer Science

Smart Systems Program

School of Engineering and Science

Prof. Dr. Michael Kohlhase

Name and title of first reviewer

Dr. Christoph Lange

Name and title of second reviewer

Date of submission: September 18, 2012

Declaration

I hereby declare that the work presented in this Master's Thesis contains original and independent results, except where sources and collaborations acknowledged, and it has not been submitted elsewhere for the conferral of a degree.

Figen Füsun Horozal
Bremen, September 18, 2012

Acknowledgments

I would like to thank my supervisors Prof. Dr. Michael Kohlhase and Dr. Christoph Lange for providing me this thesis topic and their supervision. Also, I would like to thank Dr. Andrea Kohlhase and Dr. Florian Rabe for their valuable discussions.

Abstract

Ontologies are formal representations of knowledge from various fields that range from artificial intelligence to semantic web and biomedical informatics. They often change and evolve over time. It is crucial to manage changes when working with ontologies to prevent inconsistencies that might arise from the changes made. However, it is difficult to predict and track down consequences of changes in large ontologies.

Changes and their impacts have been recently studied systematically in the research area called *management of change*. One of the frameworks that has been developed is a generic management of change service for the Module System for Mathematical Theories (MMT).

This thesis addresses management of change in the Web Ontology Language (OWL). Our goal is to investigate whether generic MMT services can be applied to specific domains, specifically, to apply the generic MMT management of change (MMT MoC) service to OWL, and thus to provide a management of change service for OWL ontologies.

As requirements of management of change in OWL, we determine dependency relations between elements in OWL, identify change impacts and represent them. In particular, we conceptually apply change impact analysis to an ontology covering all relevant change cases, and mark change impacts as annotations in the ontology. Representing change impacts as annotations permits us to visualize change impacts in existing OWL tools like Protégé.

We implement a system to apply the MMT MoC service to OWL. It detects changes, identifies impacts of the changes, and marks impacted elements. The key idea is to translate OWL ontologies to MMT, to apply the MMT MoC service to them, and then to translate them back to OWL. Thus, the MMT-based implementation is hidden from OWL users.

Our work demonstrates that it is possible to apply the generic MMT MoC service to OWL ontologies, and thus provides management of change in OWL via MMT. Similarly, future investigations can cover applying other generic MMT services to OWL, e.g., applying MMT search engine to OWL.

Contents

Chapter 1

Introduction

Ontologies are formal representations of knowledge from various fields that range from artificial intelligence to semantic web and biomedical informatics. They often change and evolve over time to meet requirements. For example, United Nations Standard Products and Services Code (UNSPSC) [UNS11], which is a real world ontology, was updated 16 times in 8 months and each update contained between 50 to 600 changes [Hug06]. There are several factors that cause changes in ontologies. Some of those are domain changes and adaptations to different applications. The lack of adaptations to changes in ontologies may cause incompatibilities with dependent ontologies and applications. Therefore, it is crucial to manage changes when working with ontologies to prevent inconsistencies that might arise from the changes made. However, it is difficult to predict and track down consequences of changes in large ontologies.

Changes and their impacts have been recently studied systematically in the research area called *management of change*. In this area, one of the frameworks is the generic management of change service that has developed for the Module System for Mathematical Theories (MMT).

MMT (A Module system for Mathematical Theories) [RK12] is a sublanguage of the OMDoc language [Koh06]. MMT has been designed as a generic declarative language so that services for MMT can be developed and applied to specific domains. Knowledge management services that are based on MMT, e.g., interactive browsing, querying and management of change, can be applied to other languages represented in MMT.

In this thesis, we address management of change in the Web Ontology Language (OWL) [MPSP09]. It is a family of formal languages for representing ontologies in the Semantic Web. Based on Description Logics (DL) [BCM$^+$03], OWL is a formal language with precise semantics.

There are some editing tools, e.g., OilEd [DBL01], OntoView [MHSV04] and Protégé [Pro11], and some OWL diff tools, e.g., OWLDiff [kri11] and Ecco [GPS12], which provide change management supports for ontologies. However, their support for management of change for ontologies is restricted such as lack of representation of changes impacts.

Our goal is to investigate whether generic MMT services can be applied to specific domains, specifically, to apply the generic MMT management of change service to OWL, and thus to provide a management of change service for OWL ontologies.

As requirements of management of change in OWL, we *1)* determine dependency relations between elements in OWL, *2)* identify change impacts, and *3)* represent change impacts. Based on the dependency relations, we identify change impacts to determine what to expect from an automated change impact analysis for OWL. In addition, we use annotations to represent change impacts. In particular, we conceptually apply change impact analysis to a use case ontology covering all relevant change cases, and mark change impacts as annotations in the ontology.

We provide a system to apply the MMT management of change service to OWL, which detects changes, identifies impacts of the changes, and marks impacted elements. It is based on our bi-directional translation between OWL and MMT. The key idea is to translate OWL ontologies to MMT, to apply MMT management of change service to them, and then to translate them back to OWL. Thus, the MMT-based implementation is hidden from OWL users. Moreover, we improved the results of our system by providing a second approach.

Scientific Contribution of this Thesis This thesis presents a broad investigation of management of change in OWL ontologies. In particular, it provides a comprehensive analysis of dependencies between elements in OWL and changes and their impacts in OWL ontologies.

Significant contributions of this thesis are *1)* our work demonstrates that it is possible to apply the generic MMT MoC service to OWL ontologies, and thus provide management of change in OWL via MMT, and *2)* our representation of change impacts as annotations inside OWL ontologies provides visualizing change impacts to OWL users in existing OWL tools like Protégé.

Thesis Overview This thesis is organized as follows.

In chapter 2, we briefly give an introduction to OWL, OMDoc and MMT, review the state of the art of services for OWL ontologies and for OMDoc, give a brief overview of Management of Change.

In chapter 3, we discuss related work on management of change in OWL ontologies.

In chapter 4, we investigate the requirements for change impact analysis using a use case ontology, and discuss changes and their impacts in OWL ontologies. Moreover, we present our representation of changes impacts as annotations inside OWL ontologies, and illustrate our representation for each kind of change using examples from our use case ontology.

In chapter 5, we present our formalizations of OWL in the logical framework LF/Twelf, which MMT uses as meta-theory for representing OWL ontologies as MMT theories. Moreover, we present out bi-directional translation between OWL and its representation in MMT.

In chapter 6, we describe our system to apply the MMT management of change service to OWL, which detects changes, identifies impacts of the changes, and marks impacted elements. We discuss adding identifiers to axioms in OWL, detecting changes and impacts using MMT, marking impacted elements. Moreover, we describe how to visualize change impacts to OWL authors by existing OWL tools.

In chapter 7, we draw conclusions on the work we have presented in this thesis, discuss our work, and mention future work.

Chapter 2

Preliminaries

2.1 Web Ontology Language

The **Web Ontology Language** (OWL) [DS04, MPSP09] is a family of formal languages for representing ontologies in the Semantic Web. It has been developed by the World Wide Web Consortium (W3C) web ontology working group. It is designed to be interpreted by computers to process information on the web. It is an expressive language based on Resource Description Framework (RDF) and RDF Schema (RDFS). In addition, it includes features from several families of representation languages, in particular from Description Logics. The Web Ontology Language defines classes, individuals, properties and their relationships to represent knowledge. A class is a set of individuals sharing a common attribute, and a property is a relationship between individuals or from individuals to datatypes.

There are two versions of the Web Ontology Language currently: OWL Web Ontology Language [DS04] and OWL 2 Web Ontology Language [MPSP09]. The OWL Web Ontology Language was published in 2002, and it was extended to the OWL 2 Web Ontology Language in 2008 by adding several new features.

2.1.1 OWL Web Ontology Language

OWL Web Ontology Language (OWL 1) [DS04] includes three sublanguages that are OWL Lite, OWL DL and OWL Full with increasing level of expressiveness. OWL Full was developed by extending of OWL DL, which is an extension of OWL Lite.

OWL Lite OWL Lite is a minimal useful subset of OWL 1 language features. It corresponds to the SHIF fragment of Description Logics with data types and

data values.

OWL DL OWL DL includes all features of the sublanguage OWL Lite. It corresponds to the SHOIN fragment of description logics with data types and data values. In addition to the features in the OWL Lite, OWL DL provides different features, for details see [DS04].

OWL Full OWL Full contains all features in the sublanguage OWL DL, which are all features of OWL 1.

2.1.2 OWL 2 Web Ontology Language

OWL 2 Web Ontology Language (OWL 2) [MPSP09] was evolved from OWL 1 by adding several new features. Three syntactic categories for representing knowledge in OWL 2 are entities, expressions and axioms. *Entities* are the basic elements of an ontology such as classes, properties and individuals, which are identified by IRIs. *Expressions* are complex elements formed from combinations of entities in an ontology. *Axioms* are statements that are asserted to be true in a domain of interest.

New Features in OWL 2 The new features in OWL 2 offer new expressivity with features such as property chains, richer data types and data ranges, qualified cardinality restrictions, asymmetric, reflexive and disjoint properties and enhanced annotation properties. For details of the new features in OWL 2 see [MPSP09].

OWL 2 Web Ontology Language defines three sublanguages called OWL 2 EL, OWL 2 QL, OWL 2 RL, which are also called OWL 2 Profiles [MGH$^+$09].

OWL 2 EL The sublanguage OWL 2 EL is based on Description Logic EL++ [BBL05]. It concentrates on terminological expressivity that are sufficient to express large ontologies.

OWL 2 QL The sublanguage OWL 2 QL is based on Description Logics DL-Lite [CLLR07]. It can be used for applications where access to data directly is necessary or useful by using relational queries.

OWL 2 RL The sublanguage OWL 2 RL is designed with rule-based implementations. It can be used for applications where scalable reasoning is implemented using a standard Rule Language.

OWL 2 DL and OWL 2 Full There are two ways of assigning meaning to ontologies called Direct Model-Theoretic semantics [MPSG09] and RDF-Based semantics [Sch09]. OWL 2 ontologies where the Direct Model-Theoretic semantics or the RDF-Based semantics is used for interpreting are called OWL 2 DL or OWL 2 Full, respectively.

2.1.3 OWL/XML Syntax

OWL/XML syntax is one of the several concrete syntaxes available to write OWL ontologies. We write our examples in OWL/XML syntax in this thesis. It was invented as a concrete representation format for OWL ontologies, because it is a regular and simple XML format. In addition, it has many advantages such as it is easy to write and parse.

Figure2.1 illustrates a simple example of an OWL 2 ontology in OWL/XML syntax, which represents the fact that Mary is John's wife. In this example, we have two individual declarations: `John` and `Mary`, and an object property declaration `hasWife`, and an object property assertion axiom that states `John` is related to `Mary` via `hasWife`.

2.2 Services for OWL

Ontology editors are used for creating and manipulating ontologies. There are various widely used editing tools such as Protégé, NeOn Toolkit [NeO11] and SWOOP [KPS+05]. They offer a wide and varied range of capabilities such as supporting multiple ontologies, using various OWL syntaxes to render ontologies and consistency checking by integrated OWL reasoners.

Protégé is a widely used Java based ontology and knowledge development tool. It supports exporting ontologies into various formats such as RDF(S) and OWL. In addition, it particularly enables building ontologies in OWL.

NeOn Toolkit is an open source ontology editor based on Eclipse platform. It has a strong emphasis on OWL 2 and especially used for big projects such as multi-modular ontologies and ontology integration.

Furthermore, SWOOP is also an open source ontology editor implemented in Java. It supports creating, editing, and debugging OWL ontologies.

```
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
  xml:base="http://example.com/owl/families/"
  ontologyIRI="http://example.com/owl/families/">

  <Declaration>
    <NamedIndividual IRI="John"/>
  </Declaration>

  <Declaration>
    <NamedIndividual IRI="Mary"/>
  </Declaration>

  <Declaration>
    <ObjectProperty IRI="hasWife"/>
  </Declaration>

  <ObjectPropertyAssertion>
    <ObjectProperty IRI="hasWife"/>
    <NamedIndividual IRI="John"/>
    <NamedIndividual IRI="Mary"/>
  </ObjectPropertyAssertion>

</Ontology>
```

Figure 2.1: An example OWL 2 ontology in OWL/XML syntax

2.3 OMDoc Markup Language

The OMDoc (Open Mathematical Documents, [Koh06]) format is a content markup language for mathematical documents. It extends the web standards content MathML [CIM10] and OpenMath [Ope09] by a semantics centered representation of mathematical knowledge. OMDoc can represent the structure of mathematical theories, i.e., elements in a theory like definitions, theorems, proofs, etc.

In addition, OMDoc uses XML [CIM11] as its markup format and in particular, it has been developed for internet applications.

OMDoc represents mathematical knowledge in three levels:

- *Theory level* declares theories and theory morphisms.
- *Statement level* declares symbols, definitions, proofs, etc.
- *Object level* declares mathematical expressions (formulae).

Object level expressions in OMDoc are *OpenMath objects*. *OpenMath objects* (om:OMOBJ) are formed by *variables* (om:OMV), *symbols* (om:OMS), *applications* (om:OMA), which apply an operator to an OpenMath object, and *bindings* (om:OMBIND), which bind a list of variables in an OpenMath object.

2.4 The MMT Language

MMT (A Module system for Mathematical Theories) [RK12] is a sublanguage of the OMDoc language. It implements a fragmant of the OMDoc format. In addition, it is a foundation-independent, modular and scalable language for formal mathematics.

MMT syntax can be classified in three levels: *module* level, *symbol* level and *object* level.

- The module level declares *theories* and *theory morphisms*.
- The symbol level declares *constants*.
- The object level consists of OpenMath expressions.

Theories contain constant declarations, which correspond to statement level declarations in OMDoc. *Constants* may have components, i.e., *types* or *definitions*, which correspond to OpenMath objects. Additionally, theories, constants and components are identified by URIs.

MMT was designed as a generic declarative language so that services for services for MMT can be developed genericly and applied to specific domains. Knowledge management services that are based on MMT, e.g., interactive browsing, querying and management of change, can be applied to other languages represented in MMT. For instance, a generic theorem search engine have been applied to Mizar [MRMJ12].

2.5 Services for MMT

In this section, we describe MMT services that we have used in our work.

The MMT management of change (MMT MoC) service [IR12] is a generic management of change service based on the MMT language. It is implemented in the MMT API [MMT]. It provides semantic differencing, dependency analysis, and change impact analysis and propagation.

MMT Diff detects four kinds of changes and identifies them as add and delete constants, rename constants, and update components.

MMT Dependency Analysis distinguishes two types of dependencies: syntactic and semantic dependencies, which are indexed by MMT.

MMT Change Impact Analysis and Propagation identifies change impacts and marks them as impacted components, and propagetes renames automatically.

Since the MMT MoC is implemented as a part of the MMT API [MMT], it can be applied to any language represented in MMT.

2.6 Management of Change

Documents are produced, evolved and changed continuously in various areas such as business, education and research. According to Madhavji [Mad92] the process of change consists of the following steps:

1. First, identify the need to make a change to an item in the environment (Identification),

2. then acquire adequate change related knowledge about the item (Change classification),

3. assess the impact of a change on other items in the environment (Dependency classification),

4. select or construct a method for the process of change (Rules),

5. make changes to all the items and make their inter-dependencies resolved satisfactorily (Adjustment),

6. record the details of the changes for future reference, and

7. release the changed item back to the environment (Version Control).

Documents are often related to other documents. Therefore, changing one document may require adaptations to other documents. However, change management, propagation of changes and identification of the resulting necessary adaptations and adjustments are highly restricted. Authors need systems to predict change impacts (change impact analysis) and to adapt the impacted items (change management) in order to avoid incompatibility. Current research on management of change improves maintainability and reduces maintenance efforts.

Management of change is widely used in science, technologies, engineering and mathematics. It has been extended to collections of semi-formal documents

and integrated it into a semantic publishing system for mathematical knowledge [ADD+11]. In addition to this, it has been shown that the semantic annotations in flexiformal documents driving the machine-supported interaction with documents can support semantic impact analyses at the same time.

Furthermore, a management of change functionality has been integrated into an active document management system. Semantic relations are used in the system in order to propagate change impacts and to keep source modules consistent. The approach is based on impact analysis by using systems of graph rewriting. However, the integration is limited to a single-user mode of operation. In addition, propagating changes by other users and multiple users working on different branches are not supported yet.

Change management on semi-structured documents provides maintaining heterogeneous collections of semi-structured documents [MÏ0]. It facilitates and improves collaboration on large document collections and also improves information consistency, reuse and distribution. It enables a framework that supports the declarative specification of semantic annotation and propagation rules.

Chapter 3

Related Work: Management of Change in OWL

Management of change in OWL ontologies is partly supported by tools such as OilEd, OntoView and Protégé. Below we give an overview for each of them.

OilEd OilEd [DBL01] is an open source editor that supports OWL ontologies and was developed at the University of Manchester. It provides a simple editing environment with enough functionality to build ontologies and keep them consistent by using the FaCT reasoner [FaC11]. In addition, it writes all activities of the tool to a log file. However, it does not support undo/redo functionality or integration and versioning of ontologies.

OntoView OntoView [MHSV04] is a web-based change management system for ontologies that was designed to compare ontology versions in RDF-based languages. It focuses on finding and representing changes between concepts in different versions of ontologies. It shows which definitions of ontological concepts or properties have changed and what kind of change occurred. When a user wants to make a change, the system provides information about consequences of the change. Furthermore, it maintains links between versions and variants specifying relations and updates between the versions of ontologies together with transformation between them. In addition, it provides support for identification of ontologies and the analysis of possible effects of changes. However, it does not allow undoing a change.

Protégé Protégé [Pro11] is a knowledge development tool for ontologies. Change facilities that are supported by Protégé are undo/redo, command history, ontology

version archiving, PROMPT and logging.

Undo/Redo functionality allows users to easily correct mistakes by reversing the last action performed and undoing the last Undo action, respectively. Command history menu item records each of change actions. Ontology version archiving provides an interface to manage different versions of an ontology. It also allows users to save the current version of an ontology and adding comments to it, and reverting it to its previous version.

PROMPT is a plugin to the Protégé editor for managing multiple ontologies. It includes tools for interactive ontology merging (iPROMPT), graph-based mapping for finding similarities between ontologies (AnchorPROMPT), factoring out semantically independent subontologies (PROMPTFactor), ontology versioning and creating views of an ontology (PROMPTDiff), and ontology library maintenance.

iPROMPT is a tool for interactive ontology merging. It integrates ontologies from different sources to create a new ontology including information from the sources. In addition, it helps users to merge ontologies by suggesting what should be merged, identifying inconsistencies and problems, and suggesting strategies to resolve them.

AnchorPROMPT is an ontology alignment tool for defining a mapping between concepts in two ontologies by finding pairs of related concepts. It analyzes a graph representing ontologies on a large scale. PROMPTFactor is a tool for extracting a part of an ontology into a new subontology. It provides specifying all concepts required in the subontology.

PROMPTDiff is a tool for comparing versions of ontologies. It finds a structural diff between two versions of the same ontology. It includes an algorithm consisting of two parts which are an extensible set of heuristic matchers and combining results of the matchers to produce a structural diff between two versions. It identifies changes such as moving classes, adding or deleting a tree of classes, shows the comparison results to the user and enables the user to accept or reject the changes between versions. Additionally, it provides two kinds of views to the structural difference of two ontologies: *Tree view* and *Table view*.

- The Tree view displays structures of ontologies and highlights changes. It shows a class tree for an ontology with new frames underlined, deleted frames crossed out, moved and changed frames shown in different fonts and colors. It also provides a table comparing the current frame to its old version. In addition, it enables users to accept or reject the changes.

- The Table view enables users to save change records as a text file. It displays a list of frames from the two versions with additional information and

an individual diff. Moreover, logging is one of the important functionalities. It allows tracking any modification applied to ontologies. However, it does not enable fully collaborative ontology development since it is not integrated with version control systems. In addition, it does not support OWL 2 ontologies, tracking changes directly, and enabling users to add design rationale during edits. Furthermore, it does not provide comments or instance information to align classes or properties, and generalization or specialization matches.

OWL Diff Tool OWL Diff Tool [RN] is another difference tool which is available as a plugin in the latest Protégé editor. It detects and shows differences such as adding, deleting and renaming of entities and annotation changes. However, it does not give a detailed description for annotation changes.

OWLDiff OWLDiff [kri11] provides syntactic compare and merge functionality for OWL ontologies. It is a standalone application developed under the LGPL license [LGP11] at the Technical University in Prague. It compares and merges ontologies and commits a resulting file. Basic syntactic difference provides inference markup and explanations between ontologies. In addition, it provides entailment checking of different axioms.

It is combined with the Pellet reasoner to show the semantic diff between two ontologies graphically in two tree views. The system comprises two algorithms to compare ontologies. Basic ontology comparison algorithm is used to find added, deleted or changed axioms except finding out complex dependencies. Besides, OWLDiff is offered as a plugin for Protégé 4.1 and the NeON toolkit.

Furthermore, it provides various visualization options such as plain axiom list, asserted frame and classified frame view. Additionally, it supports various syntax options such as Manchester syntax and Description Logics syntax. Moreover,it provides an easy SVN integration to easily compare ontologies under SVN version control. However, OWLDiff does not support imports, and explanations generated for axioms are restricted to only one explanation for each single axiom. In addition, it does not check whether the IRIs of entities and ontologies have been changed, and annotations of axioms are ignored when comparing ontologies.

Ecco Ecco [GPS12] is a recently implemented hybrid diff tool for OWL 2 ontologies. It distinguishes effectual and ineffectual changes between ontologies and presents them by categorizing. It does not support detecting annotation changes.

Chapter 4

Requirements for Management of Change in OWL

A management of change in OWL requires to detemine dependency relations between elements in OWL, to identify change impacts and to represent change impacts. We discuss these requirements in the following sections.

4.1 Dependency Relations

In this section, we discuss the dependency relations between elements in OWL 2 ontologies. The purpose of this discussion is to identify the impacts of the changes of OWL elements.

A dependency relation is a binary relation between two OWL elements. We say that *an element A depends on B*, if B occurs in A. Below, we explain the dependency relations between entities, expressions and axioms in OWL 2 ontologies.

4.1.1 Entities

Entities are basic terms that are identified by IRIs. These terms are classes, named individuals, object properties, data properties, annotation properties and data types.

Entities are declared in declaration axioms. In addition, entities can occur directly in axioms and in expressions within axioms. Therefore, expressions and axioms depend on entities that occur in them.

Expressions do not exist outside of axioms. Thus, we can say that a change of an entity can affect axioms that refer to it. As a result, if an entity has been

changed, axioms that refer to it need to be modified with respect to the change.

Below, we identify those OWL expressions and axioms in which entities occur in order to determine the impacts of a change of an entity.

Class is a set of individuals that share a common property. Classes occur in declaration axioms (declaration of a class). In addition, they can occur in class expressions and axioms as listed below.

- Class Expressions: `Class`, `ObjectIntersectionOf`, `ObjectUnionOf`, `ObjectComplementOf`, `ObjectSomeValuesFrom`, `ObjectAllValuesFrom`, `ObjectMinCardinality`, `ObjectMaxCardinality`, and `ObjectExactCardinality`.
- Class Axioms: `SubClassOf`, `EquivalentClasses`, `DisjointClasses` and `DisjointUnion`.
- Object Property Axioms: `ObjectPropertyDomain` and `ObjectPropertyRange`.
- Data Property Axioms: `DataPropertyDomain`.
- Keys: `HasKey`.
- Assertions: `ClassAssertion`.

Named Individual is an instance of a class. Named individuals occur in declaration axioms (declaration of a named individual). In addition, they can occur in the following class expressions and axioms.

- Class Expressions: `ObjectIntersectionOf`, `ObjectUnionOf`, `ObjectComplementOf`, `ObjectOneOf`, `ObjectSomeValuesFrom`, `ObjectAllValuesFrom`, `ObjectHasValue`, `ObjectMinCardinality`, `ObjectMaxCardinality` and `ObjectExactCardinality`.
- Class Axioms: `SubClassOf`, `EquivalentClasses`, `DisjointClasses` and `DisjointUnion`.
- Object Property Axioms: `ObjectPropertyDomain` and `ObjectPropertyRange`.
- Data Property Axioms: `DataPropertyDomain`.
- Keys: `HasKey`.

- Assertions: `SameIndividual`, `DifferentIndividuals`, `ClassAssertion`, `ObjectPropertyAssertion`, `DataPropertyAssertion`, `NegativeObjectPropertyAssertion` and `NegativeDataPropertyAssertion`.

Object Property is a relationship between individuals. Object properties occur in declaration axioms (declaration of an object property). In addition, they can occur in the following expressions and axioms.

- Class Expressions: `ObjectSomeValuesFrom`, `ObjectAllValuesFrom`, `ObjectHasValue`, `ObjectHasSelf`, `ObjectMinCardinality`, `ObjectMaxCardinality` and `ObjectExactCardinality`.
- Object Property Expressions: `ObjectProperty` and `InverseObjectProperty`.
- Declaration Axioms: `Declaration` (declaration of an object property).
- Class Axioms: `SubClassOf`, `EquivalentClasses`, `DisjointClasses` and `DisjointUnion`.
- ObjectProperty Axioms: `SubObjectPropertyOf`, `EquivalentObjectProperties`, `DisjointObjectProperties`, `InverseObjectProperties`, `ObjectPropertyDomain`, `ObjectPropertyRange`, `FunctionalObjectProperty`, `InverseFunctionalObjectProperty`, `ReflexiveObjectProperty`, `IrreflexiveObjectProperty`, `SymmetricObjectProperty`, `AsymmetricObjectProperty` and `TransitiveObjectProperty`.
- DataProperty Axioms: `DataPropertyDomain`.
- Keys: `HasKey`.
- Assertions: `ClassAssertion`, `ObjectPropertyAssertion` and `NegativeObjectPropertyAssertion`.

Data Property is a relationship between individuals and data values. Data properties occur in declaration axioms (declaration of a data property). In addition, they can occur in class expressions and axioms as listed below.

- Class Expressions: `DataSomeValuesFrom`, `DataAllValuesFrom`, `DataHasValue`, `DataMinCardinality`, `DataMaxCardinality` and `DataExactCardinality`.

- Class Axioms: `SubClassOf`, `EquivalentClasses`, `DisjointClasses` and `DisjointUnion`.
- ObjectProperty Axioms: `ObjectPropertyDomain` and `ObjectPropertyRange`.
- Data Property Axioms: `SubDataPropertyOf`, `EquivalentDataProperties`, `DisjointDataProperties`, `DataPropertyDomain`, `DataPropertyRange` and `FunctionalDataProperty`.
- Keys: `HasKey`.
- Assertions: `ClassAssertion`, `DataPropertyAssertion` and `NegativeDataPropertyAssertion`.

Data Type is a classifier that identifies one kind of data. Data types can occur in declaration axioms. In addition, they can occur in class expressions and axioms shown below.

- Class Expressions: `DataSomeValuesFrom`, `DataAllValuesFrom`, `DataHasValue`, `DataMinCardinality`, `DataMaxCardinality` and `DataExactCardinality`.
- Class Axioms: `SubClassOf`, `EquivalentClasses`, `DisjointClasses` and `DisjointUnion`.
- Object Property Axioms: `ObjectPropertyDomain` and `ObjectPropertyRange`.
- Data Property Axioms: `DataPropertyDomain` and `DataPropertyRange`.
- Datatype Definitions: `DatatypeDefinition`
- Keys: `HasKey`.
- Assertions: `ClassAssertion`, `DataPropertyAssertion` and `NegativeDataPropertyAssertion`.

Annotation Property is used to annotate an ontology, an axiom or an IRI. They can occur in all axioms.

4.1.2 Expressions

Expressions are complex elements formed from combinations of entities. They are grouped as *class expressions*, *object property expressions* and *data property expressions*.

Class expressions define sets of individuals by specifying conditions on the properties of individuals. Object property expressions are formed by object properties and inverse object properties. Data property expressions consist of only data properties.

Expressions can occur directly in axioms and in expressions within axioms. Therefore, expressions and axioms depend on expressions that occur in them. However, since expressions do not exist outside of axioms, we can say that a change of an expression implies a change of the axioms in which that expression occurs. As a result, we need to discuss impacts of a change of an axiom, which we explain in the next subsection.

4.1.3 Axioms

Axioms are statements that are asserted to be true in a domain of interest. They are classified as declaration axioms (declaration of entities), class axioms, object property axioms, data property axioms, data type definitions, keys, assertions, and annotation axioms. They are grouped and listed as follows:

- Declaration Axioms: `Declaration`
- Class Axioms: `SubClassOf`, `EquivalentClasses`, `DisjointClasses` and `DisjointUnion`.
- Object Property Axioms: `SubObjectPropertyOf`, `EquivalentObjectProperties`, `DisjointObjectProperties`, `InverseObjectProperties`, `ObjectPropertyDomain`, `ObjectPropertyRange`, `FunctionalObjectProperty`, `InverseFunctionalObjectProperty`, `ReflexiveObjectProperty`, `IrreflexiveObjectProperty`, `SymmetricObjectProperty`, `AsymmetricObjectProperty` and `TransitiveObjectProperty`.
- Data Property Axioms: `SubDataPropertyOf`, `EquivalentDataProperties`, `DisjointDataProperties`, `DataPropertyDomain`, `DataPropertyRange` and `FunctionalDataProperty`.
- Datatype Definitions: `DatatypeDefinition`.

- Keys: `HasKey`.
- Assertions: `SameIndividual`, `DifferentIndividuals`, `ClassAssertion`, `ObjectPropertyAssertion`, `NegativeObjectPropertyAssertion`, `DataPropertyAssertion` and `NegativeDataPropertyAssertion`.
- Annotation Axioms: `AnnotationAssertion`, `SubAnnotationPropertyOf`, `AnnotationPropertyDomain` and `AnnotationPropertyRange`.

Axioms occur in neither entities, expressions nor axioms. Therefore, entities and expressions do not depend on axioms. We know that expressions do not appear outside of axioms. Thus, we can say that a change of an axiom may have an impact on entities that occur in the axiom. As a result, if an axiom has been changed, entities that the axiom refers to may need to be changed with respect to the change. In order to find possible impacts of a change of an axiom, we identify entities that exist in axioms.

Declaration axioms, class axioms, object property axioms, data property axioms, keys and assertions can refer to all entities. In addition, datatype definitions and annotation axioms can refer to only datatype and annotation properties and only annotations properties, respectively.

Figure4.1 illustrates an example for occurrences of OWL 2 elements in OWL/XML syntax. It represents the fact that a mother is a woman who has at least one child.

In this example, a class expression that consists of a class `Mother` and a class expression `ObjectIntersectionOf` directly occur in a class axiom `EquivalentClasses`. In addition, the class expression `ObjectIntersectionOf` includes another class expression that consists of a class `Woman` and a class expression `ObjectSomeValuesFrom`. Moreover, the class expression `ObjectSomeValuesFrom` comprises an object property expression made of an object property `hasChild` and a class expression contains only a class `Person`.

4.2 Changes and their Impacts

In this section, we discuss changes and their impacts in OWL 2 ontologies to determine what is expected from an automated change impact analysis for OWL. In particular, we conceptually apply OWL change impact analysis on an example OWL ontology that covers OWL 2 features and all relevant change cases.

Ontologies can be modified according to the needs of OWL users. We consider four types of changes that can be applied to OWL ontologies: *deletion*, *addition*,

```
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
  xml:base="http://example.com/owl/families/"
  ontologyIRI="http://example.com/owl/families/">

  <Declaration>
   <Class IRI="Person"/>
  </Declaration>

  <Declaration>
   <Class IRI="Woman"/>
  </Declaration>

  <Declaration>
   <Class IRI="Mother"/>
  </Declaration>

  <Declaration>
   <ObjectProperty IRI="hasChild"/>
  </Declaration>

 <EquivalentClasses>
  <Class IRI="Mother"/>
  <ObjectIntersectionOf>
   <ObjectSomeValuesFrom>
    <ObjectProperty IRI="hasChild"/>
    <Class IRI="Person"/>
   </ObjectSomeValuesFrom>
   <Class IRI="Woman"/>
  </ObjectIntersectionOf>
 </EquivalentClasses>

</Ontology>
```

Figure 4.1: An example of occurrences of OWL 2 elements in OWL/XML syntax

renaming, and *updating*. Note that any other change can be expressed as a combination of these changes. For example, consider merging together two concepts A and B to yield C. This can be expressed as deleting A and B and adding C.

Since the elements in ontologies, e.g., entities, expressions and axioms, are related to each other via the dependency relations we discussed in Section 4.1, a change of an element can affect other elements in an ontology, and as a result, those elements need to be modified with respect to the changed element. Therefore, when an ontology is changed, it is necessary to identify changes and compute their impacts in the ontology.

In order to discuss changes and their impacts, we wrote an example ontology in OWL/XML syntax, see Appendix A, applied each kind of change we mentioned above to elements in the ontology and determined the impacts of the changes. We wrote the example ontology as an extension of the family ontology [HKP+09].

In our example ontology, there are two families: John and Mary are married

with three children, and their daughter Liz is also married with one daughter. The name of John and Mary's son is Tom, who is a teenage. In addition, their daughter Amy, who is elder than Tom, is a university student, and the name of her boyfriend is Bob. Max is Liz's husband, and their daughter Alice is 2 years old.

We applied all four kinds of changes separately to our example ontology and for each change, we saved the modified version of the ontology in a separate OWL file. In other words, we generated four versions of our example ontology, respectively by only renaming, by only deleting, by only adding, and by only updating elements of the ontology.

Moreover, we applied all changes, which we initially applied separately, together to our example ontology, and saved the modified version of the ontology in a separate OWL file.

We put all these OWL documents in `https://svn.kwarc.info/repos/MMT/src/mmt-owl/ExtendedFamilyOntology`.

This approach gives us an opportunity to discuss impacts of each change kind separately and impacts of all change kinds together.

Below, we explain the four different kinds of changes and their impacts when applied separately as well as together.

4.2.1 Renaming

The change *renaming* can only be applied to entity declarations in an ontology. Therefore, we renamed only entities in our example ontology, and saved the modified version as a new version.

We applied a renaming to an entity of each kind. In addition, we identified all impacts of the renamed entities in the ontology.

For example, let us consider renaming the individual `John` to `James`.

The `DataPropertyAssertion` axiom, shown in Figure 4.2, states that John is 51 years old, and refers to the individual `John`. Therefore, this axiom needs to be modified with respect to the renaming. In other words, we say that the renaming has an impact on this axiom.

```
<DataPropertyAssertion>
 <DataProperty IRI="hasAge"/>
 <NamedIndividual IRI="John"/>
 <Literal datatypeIRI="xsd:integer">51</Literal>
</DataPropertyAssertion>
```

Figure 4.2: An impact of renaming an entity

We identified all impacts of this renaming: other axioms that refer to the individual are similarly affected.

In contrast, the `EquivalentClasses` axiom, shown in Figure 4.3, refers to a class `JohnsChildren` that represents a class whose members are John's children. The name of the class `JohnsChildren` contains the name of the individual `John` as a substring. It would be intuitive to rename the class `JohnsChildren` as well. One could use substring matching to capture such cases. Note that substring matching can also give results that should not be impacted. For example, assume an individual `Chil` is renamed, and there is a class `Children`. Obviously, substring matching does not work for this particular example. Nevertheless, OWL users can decide which results of substring matching to consider.

```
<EquivalentClasses>
 <Class IRI="JohnsChildren"/>
 <ObjectOneOf>
  <NamedIndividual IRI="Liz"/>
  <NamedIndividual IRI="Tom"/>
  <NamedIndividual IRI="Amy"/>
 </ObjectOneOf>
</EquivalentClasses>
```

Figure 4.3: An impact of renaming an entity

As a result, if an entity has been renamed, axioms that refer to the entity need to be modified with respect to the renamed entity. In other words, renaming an entity has an ***impact*** on axioms that refer to it.

4.2.2 Deletion

Entity declarations and axioms can be deleted in an ontology. Therefore, we deleted entities and axioms in our example ontology, and saved the modified ontology as a new version. In addition, we identified all impacts of the deleted elements in the ontology.

First, we applied a deletion to an entity of each kind in our example ontology, and then determined the impacts of the deleted entities.

For example, let us consider deleting the object property `hasAncestor`. The `TransitiveObjectProperty` axiom, shown in Figure 4.4, states that the object property `hasAncestor` is transitive, and refers to `hasAncestor`. Therefore, this axiom needs to be modified with respect to the deletion since the object property `hasAncestor` does not exist anymore. In other words, the deletion has an impact on the axiom.

```
<TransitiveObjectProperty>
 <ObjectProperty IRI="hasAncestor"/>
</TransitiveObjectProperty>
```

Figure 4.4: An impact of deleting an entity

The other impacts of a deleted entity are similarly those axioms that refer to the entity.

As a result, if an entity has been deleted, axioms that refer to the entity need to be modified with respect to the deleted entity. In other words, deleting an entity has an ***impact*** on axioms that refer to it.

Second, we applied a deletion to an axiom in our example ontology, and then identified impacts of the deleted axiom. For example, let us remove the assertion axiom, shown in Figure 4.5, from our ontology. It states that `Liz` and `Liza` are the same individuals.

```
<SameIndividual>
 <NamedIndividual IRI="Liz"/>
 <NamedIndividual IRI="Liza"/>
</SameIndividual>
```

Figure 4.5: An example of deleting an axiom

The deleted axiom does not have any direct effect on the entities, expressions or the axioms in our example ontology since it does not occur in those elements. Nevertheless, deleting an axiom means removing a statement asserted to be true about the entities in the axiom. Therefore, elements in the ontology may need to be modified with respect to the entities in the deleted axiom.

Thus, we identify declarations of the entities as *possibly impacted elements*. As a result, if an axiom has been deleted, entities that the axiom refers to may need to be modified with respect to the deleted axiom. In other words, deleting an axiom has a ***possible impact*** on entities that the axiom refers to.

Note that we give only one example for deleting an entity and an axiom, and their impacts here. Examples for deleting entities and axioms, and their impacts in the ontology can be found at `https://svn.kwarc.info/repos/MMT/src/mmt-owl/ExtendedFamilyOntology`in detail.

4.2.3 Addition

Entity declarations and axioms can be added to an ontology. Therefore, we added entities and axioms to the ontology, and saved the modified ontology as a new

version. We also identified all impacts of the added elements in the ontology.

First, we added an entity of each kind to the ontology, and then identified impacts of the added entities. For example, let us add a class `Student`, shown in Figure 4.6.

```
<Declaration>
 <Class IRI="Student"/>
</Declaration>
```

Figure 4.6: An example of adding an entity

This addition does not affect any of the elements since there is no element that refers to the added class in the ontology. As a result, adding an entity does not have any impact on elements in an ontology.

Second, we added axioms to our example ontology, and then identified the impacts of the added axioms. For instance, let us add the following `ClassAssertion` axiom shown in Figure 4.7, which states that Bob has no children.

```
<ClassAssertion>
 <Class IRI="ChildlessPerson"/>
 <NamedIndividual IRI="Bob"/>
</ClassAssertion>
```

Figure 4.7: An example of adding an axiom

The added axiom does not have any direct effect on elements in the ontology. However, the added axiom may cause an inconsistency in the ontology. For example, there may already be an axiom stating that Bob is a parent in the ontology. In this case, adding this axiom would definitely lead to an inconsistency, since each member of the class `ChildlessPerson` is defined as a person who is not a parent, shown in Figure 4.8, in the ontology. Therefore, the entities that the added axiom refers to may have to be modified with respect to the added axiom. Thus, we identify the entities as *possibly impacted* elements. In other words, we say that the addition has a ***possible*** impact on the entities.

As a result, if an axiom has been added, entities that the axiom refers to may need to be modified with respect to the added axiom. In other words, adding an axiom has a possible impact on entities that the axiom refers to.

4.2.4 Updating

Axioms can be updated in an ontology. We updated several axioms in our example ontology, and saved the modified ontology as a new version. In addition,

```
<EquivalentClasses>
 <Class IRI="ChildlessPerson"/>
 <ObjectIntersectionOf>
  <Class IRI="Person"/>
  <ObjectComplementOf>
   <Class IRI="Parent"/>
  </ObjectComplementOf>
 </ObjectIntersectionOf>
</EquivalentClasses>
```

Figure 4.8

we identified all impacts of the updated axioms. For example, we updated the following axiom in Figure 4.9, which states that Tom is a teenager, as an axiom stating that Tom is a childless person, shown in Figure 4.10.

```
<ClassAssertion>
 <Class IRI="Teenager"/>
 <NamedIndividual IRI="Tom"/>
</ClassAssertion>
```

Figure 4.9: An example of an axiom

```
<ClassAssertion>
 <Class IRI="ChildlessPerson"/>
 <NamedIndividual IRI="Tom"/>
</ClassAssertion>
```

Figure 4.10: An example of an updated axiom

Updating the axiom does not have any impact on the elements in our example ontology. However, it may cause an inconsistency in the ontology. For example, there may already be an axiom stating that Tom has a child in the ontology. In this case, this axiom would contradict the current version of the axiom. For this reason, the entities that the current version of the axiom refers to need to be modified with respect to the update. Thus, we identify those entities as possibly impacted elements. That means the updating has a possible impact on those entities.

As a result, if an axiom has been updated, entities that the current axiom refers to may need to be modified with respect to the current axiom. In other words, updating an axiom has a ***possible impact*** on entities that the current axiom refers to.

```
<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasRelative"/>
 <NamedIndividual IRI="John"/>
 <NamedIndividual IRI="Alice"/>
</ObjectPropertyAssertion>
```

Figure 4.11: An example of impacts of multiple changes

4.2.5 Multiple Changes

We now apply all four changes we mentioned in Section 4.2.1, Section 4.2.2, Section 4.2.3 and Section 4.2.4 to our example ontology at once to discuss the impacts of different kinds of changes together.

For example, renaming the individual `John` to `James` and deleting the individual `Alice` affect the `ObjectPropertyAssertion` axiom in Figure 4.11, which states that Alice is John's relative.

Similarly, applying multiple changes to different axioms at once may have multiple impacts on entities that the axioms refers to. As a result, multiple changes can have multiple impacts on the same axiom and multiple possible impacts on the same entity declaration in an ontology.

4.3 Representing Change Impacts

In this section, we establish our requirements for representing change impacts in OWL ontologies. We discussed all possible kinds of changes and their impacts in Section 4.2. As a next step, we need to represent them in a way that shows what has changed and what the impacts of the changes are. The representation needs to

1. be designed in a way that change impacts can be displayed to users in any OWL editor or diff tool, and

2. distinguish different impacts from each other.

One way to satisfy these two requirements is to use annotations. We choose to represent change impacts by writing them as annotations inside ontologies themselves. This representation brings up further requirements, which we explain below.

4.3.1 Change Impacts as Annotations

We can represent change impacts as annotations inside an ontology itself. First of all, we need to refer to changed elements in order to represent their impacts on elements in an ontology. We also need to refer to changed entities and changed axioms that have effects on axioms and entities in the ontology, respectively. Note that entities are identified by IRIs, but axioms do not necessarily have identifiers in OWL. Therefore, one of the requirements is to add identifiers to axioms in an ontology.

We can add an identifier to an axiom as an annotation. We define an annotation property with an IRI `http://omdoc.org/identifier#id` and a literal of type string for an identifier of an axiom.

For example, the axiom in Figure 4.12 states that John is 51 years old and it has the identifier ax-1157570453. Thus, if this axiom has been changed, we can refer to it by its axiom identifier.

```
<DataPropertyAssertion>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
  <Literal datatypeIRI="xsd:string">ax-1157570453</Literal>
 </Annotation>
 <DataProperty IRI="hasAge"/>
 <NamedIndividual IRI="John"/>
 <Literal datatypeIRI="xsd:integer">51</Literal>
</DataPropertyAssertion>
```

Figure 4.12: An example of adding an identifier to an axiom

We add identifiers to axioms in our extended family ontology, see `https://svn.kwarc.info/repos/MMT/src/mmt-owl/ExtendedFamilyOntology`

Next, we need to define a special annotation structure to distinguish annotations to change impacts from other kinds of annotations.

We define an annotation that consists of an annotation property with an IRI in the `http://omdoc.org/impact#` namespace and a literal of type string explaining a change and its impact. Thus, we can distinguish these annotations from other annotations in an ontology. In order to distinguish different impacts from each other, we need to define an annotation property for each change type.

In addition to this, as we mentioned in Section 4.2, a change of an entity declaration can affect axioms that refer to it (impact), and a change of an axiom may have an impact on entity declarations that the axiom refers to (possible impact). Therefore, we define annotation properties with the IRIs shown in Table 4.1 for impacts and Table 4.2 for possible impacts, due to the four kinds of changes, i.e., deletion, addition, renaming, and updating in ontologies.

Change Types	Identifiers for Impacts
Deletion	`http://omdoc.org/impact#impactByDelete`
Renaming	`http://omdoc.org/impact#impactByRename`

Table 4.1: Identifiers for impacts with respect to change types

Change Types	Identifiers for Possible Impacts
Deletion	`http://omdoc.org/impact#possibleImpactByDelete`
Addition	`http://omdoc.org/impact#possibleImpactByAdd`
Updating	`http://omdoc.org/impact#possibleImpactByUpdate`

Table 4.2: Identifiers for possible impacts with respect to change types

Thus, OWL tools can display impacts of different kind of changes according to the needs of OWL users. For example, we can only display impacts of deleted elements in an ontology, or we can show each type of impact in a different color.

Furthermore, we need to represent an impact of a change as an axiom annotation. We know that an entity can occur in an axiom directly or in an expression that exists within an axiom, see Section 4.1. Although an occurrence of an entity or an expression in an axiom is affected by a change, we need to annotate the whole axiom rather than the occurrence itself, since we can annotate axioms but not entities or expression in an axiom. But, an axiom is uncomplicated and small enough for OWL users that we expect them to understand a change impact in an axiom from its axiom annotation.

Moreover, we need to add an annotation for each impact on an axiom. For example, if an axiom is affected by deleting an entity declaration and renaming another entity, we need to add two separate annotations that state impacts of those changes on the axiom.

We explain how to annotate elements that are impacted by each kind of changes and all kinds of changes together in the following sections.

4.3.2 Renaming

After identifying impacts of each renamed entity, we manually annotated the impacted axioms in our example ontology with respect to the renamed entities, and saved the modifications as a new version of the ontology, see `https://svn.kwarc.info/repos/MMT/src/mmt-owl/ExtendedFamilyOntology`.

For example, as we discussed in Section 4.2.1, when the individual `John` is renamed to `James` it has an impact on the `DataPropertyAssertion` axiom in

Figure 4.2. We add an annotation to this axiom by indicating that "This axiom has been impacted by renaming individual John." as shown in Figure 4.13.

```
<DataPropertyAssertion>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#impactByRename"/>
  <Literal datatypeIRI="xsd:string">This axiom has been impacted by renaming individual John.
  </Literal>
 </Annotation>
 <Annotation>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
  <Literal datatypeIRI="xsd:string">ax−1157570453</Literal>
 </Annotation>
 <DataProperty IRI="hasAge"/>
 <NamedIndividual IRI="John"/>
 <Literal datatypeIRI="xsd:integer">51</Literal>
</DataPropertyAssertion>
```

Figure 4.13: An example of representing an impact of renaming an entity

We give another example, where we annotate the `EquivalentClasses` axiom in Figure 4.14 that refers to the class `John`. In addition, as we explained in Section 4.2.1, it would be intuitive to rename the class `JohnsChildren` as well. This annotation may also help OWL authors to notice the impact of renaming `John` on the class `JohnsChildren`.

```
<EquivalentClasses>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#impactByRename"/>
  <Literal datatypeIRI="xsd:string">This axiom has been impacted by renaming individual John.
  </Literal>
 </Annotation>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
  <Literal datatypeIRI="xsd:string">ax163847801</Literal>
 </Annotation>
 <Class IRI="JohnsChildren"/>
 <ObjectHasValue>
  <ObjectProperty IRI="hasParent"/>
  <NamedIndividual IRI="John"/>
 </ObjectHasValue>
</EquivalentClasses>
```

Figure 4.14: An example of representing an impact of renaming an entity

4.3.3 Deletion

We manually annotate the impacted entities and axioms with respect to the deleted elements in the ontology after identifying impacts of each deleted entity, and save them as another version of the ontology, see `https://svn.kwarc.info/repos/MMT/src/mmt-owl/ExtendedFamilyOntology`

First, we annotate all impacts of deleted entities in the ontology. For example, we add an annotation to the `TransitiveObjectProperty` axiom stating that an object property hasAncestor is transitive, shown in Figure 4.4, since it is affected by deleting the object property `hasAncestor`. The annotation states that "This axiom has been impacted by deleting objectProperty hasAncestor." as shown in Figure 4.15.

```
<TransitiveObjectProperty>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#impactByDelete"/>
  <Literal datatypeIRI="xsd:string">This axiom has been impacted by deleting objectProperty hasAncestor.
  </Literal>
 </Annotation>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
  <Literal datatypeIRI="xsd:string">ax-1444772685</Literal>
 </Annotation>
 <ObjectProperty IRI="hasAncestor"/>
</TransitiveObjectProperty>
```

Figure 4.15: An example of representing an impact of deleting an entity

Second, we annotate all possible impacts of deleted axioms in the ontology. For example, we know that removing the assertion axiom stating that `Liz` and `Liza` are the same individuals, shown in Figure 4.5, from the ontology has a possible impact on the individuals `Liz` and `Liza`. Therefore, we annotate both of the individuals by indicating that "An axiom with ID ax-1711766800 that includes this entity has been deleted."', as shown in Figure 4.16.

4.3.4 Addition

As we discussed in Section 4.2.3, adding an entity does not have any impact on elements in our ontology. Therefore, we do not need to add any annotation for this case.

We manually added an annotation for each possible impact of the added axioms in the ontology, and saved the modified ontology as a new version, see `https://svn.kwarc.info/repos/MMT/src/mmt-owl/ExtendedFamilyOntology`. For example, the possible impacts of adding the `ClassAssertion` axiom in Figure 4.7,

```
<Declaration>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#possibleImpactByDelete"/>
  <Literal datatypeIRI="xsd:string">An axiom with ID ax−1711766800 that includes this entity has been
      deleted.
  </Literal>
 </Annotation>
 <NamedIndividual IRI="Liz"/>
</Declaration>

<Declaration>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#possibleImpactByDelete"/>
  <Literal datatypeIRI="xsd:string">An axiom with ID ax−1711766800 that includes this entity has been
      deleted.
  </Literal>
 </Annotation>
 <NamedIndividual IRI="Liza"/>
</Declaration>
```

Figure 4.16: An example of representing a possible impact of deleting an axiom

which statesthat Bob has no children, are on the class `ChildlessPerson` and on the individual `Bob`. Therefore, we annotate these two entities by stating that "An axiom with ID that includes this entity has been added.", shown in Figure 4.17.

```
<Declaration>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#possibleImpactByAdd"/>
  <Literal datatypeIRI="xsd:string">An axiom with ID that includes this entity has been added.
  </Literal>
 </Annotation>
 <NamedIndividual IRI="Bob"/>
</Declaration>

<Declaration>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#possibleImpactByAdd"/>
  <Literal datatypeIRI="xsd:string">An axiom with ID that includes this entity has been added.
  </Literal>
 </Annotation>
 <Class IRI="ChildlessPerson"/>
</Declaration>
```

Figure 4.17: An example of representing a possible impact of adding an axiom

4.3.5 Updating

After identifying possible impacts of each updated axiom in our example ontology, we manually annotated the entities that the current axiom refers to with respect

to the update, and save the modified ontology as a new version, see `https://svn.kwarc.info/repos/MMT/src/mmt-owl/ExtendedFamilyOntology`.

For example, updating the axiom stating that Tom is a teenager, see Figure 4.9, to an axiom stating that Tom is a childless person, see Figure 4.10, has a possible impact on the entities that the current axiom refers to. Therefore, we add an annotation stating that "An axiom with ID ax1003750178 that includes this entity has been added." for each entity as shown in Figure 4.18.

```
<Declaration>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#possibleImpactByUpdate"/>
  <Literal datatypeIRI="xsd:string">An axiom with ID ax1003750178 that includes this entity has been
      added.
  </Literal>
 </Annotation>
 <NamedIndividual IRI="Tom"/>
</Declaration>

<Declaration>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#possibleImpactByUpdate"/>
  <Literal datatypeIRI="xsd:string">An axiom with ID ax1003750178 that includes this entity has been
      added.
  </Literal>
 </Annotation>
 <Class IRI="ChildlessPerson"/>
</Declaration>
```

Figure 4.18: An example of representing a possible impact of updating an axiom

4.3.6 Multiple Changes

As we discussed in Section 4.2.5, multiple changes can have impacts and possible impacts on an ontology element. Therefore, we manually added an annotation for each impact on the elements in our example ontology, and save the modified ontology as a new version in `https://svn.kwarc.info/repos/MMT/src/mmt-owl/ExtendedFamilyOntology`.

For example, renaming the individual `John` to `James` and deleting the individual `Alice` affect the `ObjectPropertyAssertion` axiom stating that Alice is John's relative, shown in Figure 4.11. We add one annotation stating that "This axiom has been impacted by renaming individual John." for the impact of the renamed individual, and another annotation stating that "This axiom has been impacted by deleting individual Alice." to the affected axiom as shown in Figure 4.19.

```
<ObjectPropertyAssertion>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#impactByRename"/>
  <Literal datatypeIRI="xsd:string">This axiom has been impacted by renaming individual John.
  </Literal>
 </Annotation>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#impactByDelete"/>
  <Literal datatypeIRI="xsd:string">This axiom has been impacted by deleting individual Alice.
  </Literal>
 </Annotation>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
  <Literal datatypeIRI="xsd:string">ax-1498503059</Literal>
 </Annotation>
 <ObjectProperty IRI="hasRelative"/>
 <NamedIndividual IRI="John"/>
 <NamedIndividual IRI="Alice"/>
</ObjectPropertyAssertion>
```

Figure 4.19: An example of representing impacts of multiple changes

Chapter 5

Representation of OWL in MMT

5.1 Formalizing OWL in Twelf

In this section we explain our work on representing OWL in a logical framework. We use this representation as a meta theory in MMT for representing OWL ontologies as MMT theories. The particular logical framework we used for this purpose is the Twelf module system [RS09a]).

5.1.1 Twelf Module System

There are various frameworks in which formal languages and logics can be represented. They have been developed to reason about formal languages and logics, in particular to understand relationships between other languages and logics. The Edinburgh Logical Framework (LF) [HHP93] is one such framework that is based on proof theory. Twelf [PS99] is an implementation of LF, which is used as a logic programming language and as a system for formalizing mathematics.

The module system of Twelf [RS09a] is a recent extension of Twelf that builds MMT's module system features on top of Twelf. It uses a graph structure that consists of modules called *theories*, and two different kinds of links between two given theories, namely *inclusions* and *views* (also known as *theory morphisms*). An inclusion of a theory S to another theory T means that T copies the content of S and possibly adds more content on top of that. Views typically translate the content of one theory to expressions in the other theory.

An example of a theory is the following one:

```
%sig OWLBase = {
  class          : type.
  objectProperty : type.
  individual     : type.
```

}.

The name of the theory is `OWLBase`, and it declares three symbols: `class`, `objectProperty` and `individual` to be used as types.

A view between the theory `OWLBase` to some other theory maps the symbols `class`, `objectProperty`, `individual` and to expressions in the other theory. An example of a view `OWLBaseFOLEQD` from the theory `OWLBaseFOLEQD` to another theory `FOLEQD` of first-order logic is the following one, where i and o represent first-order terms and formulae, respectively:

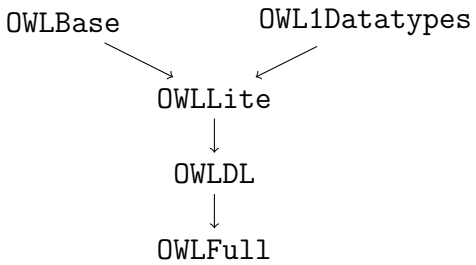
%view OWLBaseFOLEQD : OWLBase −> FOLEQD = {
 class $:= i \to o$.
 objectProperty $:= i \to i \to o$.
 individual $:= i$.
}.

Each class corresponds to a unary predicate, each property corresponds to a binary predicate and each individual corresponds to an individual in first-order logic.

5.1.2 Representing Web Ontology Language in Twelf

We formalized Web Ontology Language by encoding OWL 1 and OWL 2 in Twelf. In particular, we encoded various features of them in Twelf's module system [RS09b]. The main feature of the encoding is the use of Twelf's module system for representing different sublanguages of the two existing versions of Web Ontology Language in separate modules called theory.

OWL 1 Web Ontology Language We encode each sublanguage of OWL 1 modularly in a separate theory as illustrated on the diagram below. Each node in this diagram is a theory and edges between the nodes are inclusions.



First, we represent basic notions such as classes, object properties, data properties and individuals in a separate theory `OWLBase` by declaring types for each of

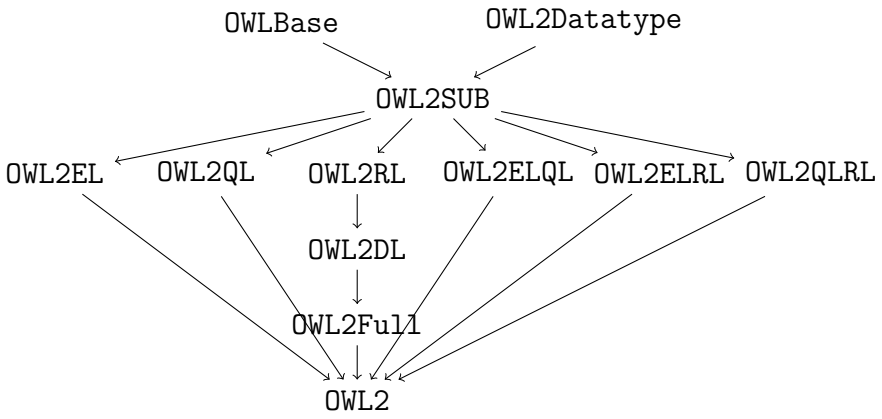
these different kinds of basic notions in OWL 1. Then we represent OWL 1 data types such as *string*, *boolean* and *integer* in another theory `OWL1Datatype`.

Encoding of OWL Lite in Twelf is based on the types we have introduced for the basic notions and the data types. Therefore, the theory `OWLLite` first includes the theories `OWL1Datatype` and `OWLBase`. In addition, it comprises encodings of all the other features of OWL Lite.

Encoding of OWL DL is based on the features of OWL Lite. Therefore, the theory `OWLDL` includes the theory `OWLLite` and consists of additional symbol declarations.

The OWL Full sublanguage contains all the features of OWL Lite. Therefore, the theory `OWLFull` includes the theory `OWLDL`. That means the theory `OWLFull` comprise the encodings of all the features in OWL 1.

OWL 2 Web Ontology Language We encode each sublanguage of OWL 2 modularly in a separate theory as illustrated in the diagram below.



First, we encode the data types of OWL 2 in the theory `OWL2Datatype`. Since OWL 2 data types consist of OWL 1 data types as well, the theory `OWL2Datatype` includes the theory `OWL1Datatype` and some extra data types such as `rational`, `real` and `dataTimeStamp`.

Then we define a theory named `OWL2SUB` including the theory `OWLBase` and `OWL2Dataype` and consisting of the common features in all the sublanguages of OWL 2.

The theory `OWL2EL`, `OWL2QL` and `OWL2RL` include the theory `OWL2SUB` and they consist of the features which are only in the sublanguage OWL 2 EL, OWL 2 QL and OWL 2 RL, respectively.

```
%read "owl2.elf".

%sig T2 = {
        %include OWL2SUB %open.

        John    : individual.
        Mary    : individual.
        hasWife : objectProperty.
        ax2     : (objectPropertyAssertion hasWife John Mary).
}.
```

Figure 5.1: An example OWL 2 ontology in LF

We define the theories `OWL2DL` and `OWL2Full` for the sublanguages of OWL 2 DL and OWL 2 Full, respectively. The theory `OWL2DL` includes the theory `OWL2RL`, and the theory `OWL2Full` includes the theory `OWL2DL`. Moreover, we define a theory named `OWL2` that includes all the theories to represent the OWL 2 Web Ontology Language.

Representing OWL Ontologies in Twelf We can now use our Twelf representation of OWL to actually represent OWL ontologies in Twelf.

Figure 5.1 illustrates the example ontology 2.1 in Twelf. We write the example ontology by using our OWL encodings in LF. Types `class`, `objectProperty` and `individual` are declared in the theory `OWLBase`, and the type constructor `ObjectPropertyAssertion` is declared in the theory `OWL2SUB` also including the theory `OWLBase`.

The example theory `T2` includes the theory `OWL2SUB` since we use the theory `OWL2SUB` as a meta theory. `John` and `Mary` are declared of type `individual` and the relation `hasWife` is declared of type `objectProperty`. Also, the object property assertion axiom `ax2` is declared by using the type constructor taking the `objectProperty hasWife` and the two individuals `John` and `Mary`.

5.2 Bi-Directional Translation between OWL and MMT

Implementation is a crucial prerequisite for applying to OWL ontologies knowledge management services such as management of change for MMT. We use our representation of OWL in the Twelf system a meta theory for the MMT representation of OWL ontologies. Moreover, the implementation is maintainable and applicable in practice.

MMT provides certain strengths such as modularity, documentation, management of change, browsing and editing. It is desired to apply these MMT services to OWL in order to solve complementary problems. However, it is not feasible to completely work on the MMT side. Therefore, we require a smooth translation between OWL and MMT in order to apply any MMT based service to OWL. A bi-directional translation provides a basis for applying MMT services to OWL. A translation from OWL to MMT and the back has to be mutually inverse, and has to cover the Web Ontology Language.

5.2.1 Translation from OWL to MMT

We present our translation from OWL to MMT in this section. We use our encodings of OWL1 and OWL2 in Twelf as a meta theory for the MMT representation of ontologies.

An ontology corresponds to a *theory* in MMT. Therefore, we translate ontologies to theories. The attributes of a *theory* are *meta*, *name* and *base*. We set the value of *meta* to the path of the meta theory and the value of *base* to the value of the corresponding ontology IRI.

An entity can occur in a declaration axiom or an expression in an axiom other than declaration in OWL. An entity in a declaration axiom is translated to a *constant* of type *symbol* (OMS) in MMT and an entity in an expression is translated to a *term* in MMT. In addition, each axiom in OWL is translated to a *constant* of type *application* in MMT.

Furthermore, an expression, a data range, and a facet restriction in OWL correspond to an *application* (OMA) of a term to a list of terms in MMT. OMA takes a function term and the list of argument terms. Therefore, each expression and each data range is translated to the corresponding *application* in MMT.

Moreover, a literal in OWL corresponds to either an *OMLiteral* or an *application* in MMT. Literals that are typedLiteral correspond to OMLiteral which can be either OMI, OMF, OMSTR or OMURI for integers, doubles, strings or URI, respectively in MMT. Therefore, each typedLiteral is translated to the corresponding OMLiteral. Literals that are stringLiteralNoLanguage and stringLiteralWithLanguage do not correspond any *Term* in MMT. Therefore, stringLiteralNoLanguage and stringLiteralWithLanguage are translated to applications(OMA) in MMT.

Finally, an annotation in OWL corresponds to a *meta datum* in MMT. Therefore, each annotation is translated to a meta datum in MMT.

5.2.2 Translation from MMT to OWL

We present our translation from MMT to OWL in this section.

A theory in an MMT document corresponds to an *ontology* in OWL. Therefore, we translate theories to ontologies. The attributes of a *theory* are *meta*, *name* and *base*. We set the value of *ontology IRI* to the *base* value of the corresponding theory.

A *constant* in an MMT document corresponds to an *axiom* in OWL. Therefore, each constant in MMT is translated to an axiom in OWL. If the type of a constant matches an MMT *symbol* (OMS) with an entity name, then we get the GlobalName of the constant and translate it to an IRI. After that, we create an entity declared by the IRI and add a *declaration axiom* for the entity to the ontology. If the type of the constant matches an MMT *application* (OMA) with a specific axiom, then we get the arguments of the application and create an axiom corresponding to the function of application.

A *Term* in an MMT document corresponds to a class expression, individual, object property expression, data property expression, data range, literal or facet restriction in OWL. Therefore, the translation of terms is analogous to the translation from OWL to MMT.

5.3 Plugin for MMT

We implemented the bidirectional translation between OWL and MMT as a plugin for MMT. The reason why we implement the plugin is that we take the advantage of using the plugin such as running the translation automatically and utilizing features of MMT. The following steps are set up to use the plugin.

1. check out the LATIN archive at `http://alpha.tntbase.mathweb.org/repos/cds`
2. create a new folder for your archive and put all your OWL files in a subfolder named `source`
3. create a manifest file which gives information about the archive. An example manifest file written for our test archive can be found at `https://svn.kwarc.info/repos/MMT/src/mmt-owl/Test/META-INF/MANIFEST.MF`
4. run MMT class `info.kwarc.mmt.api.frontend.Run` with the following JAR files on the Java class path
    - scala-library.jar

- mmt-lf.jar
- mmt-api.jar
- mmt-owl.jar
- owlapi-bin.jar

5. execute the following MMT commands

   - `archive add` path of the LATIN archive relative to the path on your computer; to register the latin archive since the bidirectional translation depends on it
   - `importer info.kwarc.mmt.owl.OWLCompiler`; to inform MMT where the compiler of the translation from OWL to MMT is
   - `archive add` the path of your archive; to register your archive
   - `archive` the id of your archive `compile`; to compile all OWL files in the source folder
   - `archive` the id of your archive `content`; to rearrange and to index MMT files for dependency analysis
   - `foundation info.kwarc.mmt.lf.LFF`
   - `archive` the id of your archive `check`

Chapter 6

Implementation of Management of Change in OWL via MMT

We provide a management of change system for OWL ontologies via MMT by implementing a system to apply the MMT management of change service to OWL. It detects changes, identifies impacts of the changes, and marks impacted elements. This system is based on our bi-directional translation between OWL and MMT. The key idea is to translate OWL ontologies to MMT, to apply MMT management of change service to them, and then to translate them back to OWL. Thus, the MMT-based implementation is transparent to OWL users.

The source code and a user manual of our system are publicly available on SVN at the following address: `https://svn.kwarc.info/repos/MMT/src/mmt-owl`

We give an overview of the workflow of our system in Section 6.1 and we describe our second approach that improves the results of the system in Section 6.2.

We describe the steps in the workflow as follows: adding identifiers to OWL axioms in Section 6.3, translating from OWL to MMT in Section 5.2.1, dependency analysis in Section 2.5, detecting changes by MMT Diff in Section 6.4, change impact analysis in Section 2.5, marking change impacts as annotations in Section 6.5, translating from MMT to OWL in Section 5.2.2, and visualizing change impacts in Section 6.6.

6.1 Workflow

We initially inform OWL users that if they add identifiers to axioms, see Section 6.3, in the previous version and the current version of an ontology before feeding them into our system, they can obtain precise results.

The work flow of our system shown in Figure 6.1 is as follows.

1. A previous version (OWL) and the current version (OWL') of an OWL ontology are taken as inputs, and identifiers are automatically added to them.

2. Both ontologies with identifiers (OWL^+) and (OWL'^+) are translated to their representations in MMT (MMT^+) and (MMT'^+) respectively.

3. Dependency analysis is applied to MMT^+ to determine occurs-in relations of the previous ontology.

4. MMT Diff is applied to compare MMT^+ with MMT'^+ to detect changes between the previous ontology and the current ontology.

5. MMT Change impact analysis is applied to the occurs-in relations and the changes to obtain impacts of the changes.

6. The change impacts are marked in MMT'^+ as meta data, which yields (MMT'^{+*}).

7. MMT'^{+*} is translated back to its representation in OWL (OWL'^{+*}).

   In addition, step 8 and step 9 are optional to users.

8. OWL'^+ and OWL'^{+*} are compared by an OWL diff tool to see only the impacts of changes on the current ontology.

9. OWL^+ and OWL'^{+*} are also compared by an OWL diff tool to see the differences between the previous ontology and the current ontology as well as the change impacts.

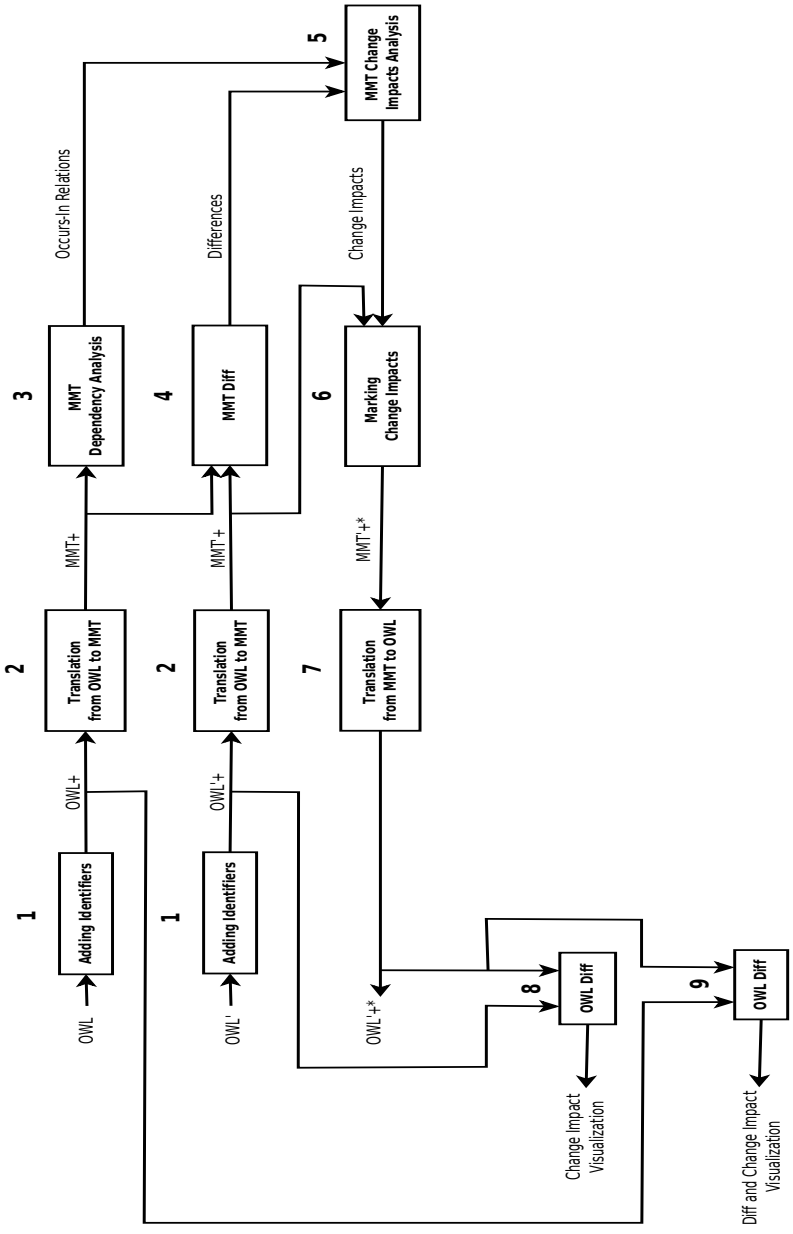


Figure 6.1: The first workflow of Management of Change in OWL via MMT

6.2 Improving Results

We develop a solution to improve the results of our system. In this way, users do not have to add identifiers to axioms to obtain more precise results. Instead, they have to apply the changes to a derived ontology with identifiers from the previous ontology, which is generated by our system.

Our system initially adds identifiers to axioms, see Section 6.3, in the previous version of an ontology, and then asks users to make their modifications on this derived ontology with identifiers. Thus, changes can be detected not only as deletion and addition operations but also as updating operations. Then, both the derived ontology with identifiers and the current ontology yielded from the changes are taken as inputs to the further processing steps.

The work flow of the second approach shown in Figure 6.2 is different from the first workflow in the following ways.

i Initially, a previous version (OWL) of an OWL ontology is taken, and identifiers are automatically added to axioms in the ontology, which returns a derived ontology with identifiers (OWL^{+}) from the previous ontology.

ii OWL^{+} is modified by users, which yields the current version of the ontology ($OWL^{+'}$).

The derived ontology with identifiers OWL^{+} and the current ontology ($OWL^{+'}$) are taken as inputs for the the same processing steps of the first work flow.

6.3 Adding Identifiers to OWL Axioms

Entity declarations are identified by IRIs, but axioms do not have identifiers in OWL ontologies. We need to add identifiers to axioms to detect changes more precisely by MMT Diff and to refer to changed axioms when we apply MMT Change impact analysis to OWL ontologies that are translated to MMT.

We define a special annotation structure to represent identifiers, see Section 4.3.1, and add identifiers to axioms as annotations.

We know that entity declarations and axioms in OWL ontologies correspond to constants in MMT, see Section 5.2.1. MMT Diff needs to compare two constants which have the same name to be able to figure out whether there is no change between the two constants or the change is an update. Otherwise, both cases are detected as delete and add operations. Therefore, there are two requirements: *1)* the name of the constants must be unique and *2)* the same constants must be given the same name in two versions of MMT documents. Thus, a special

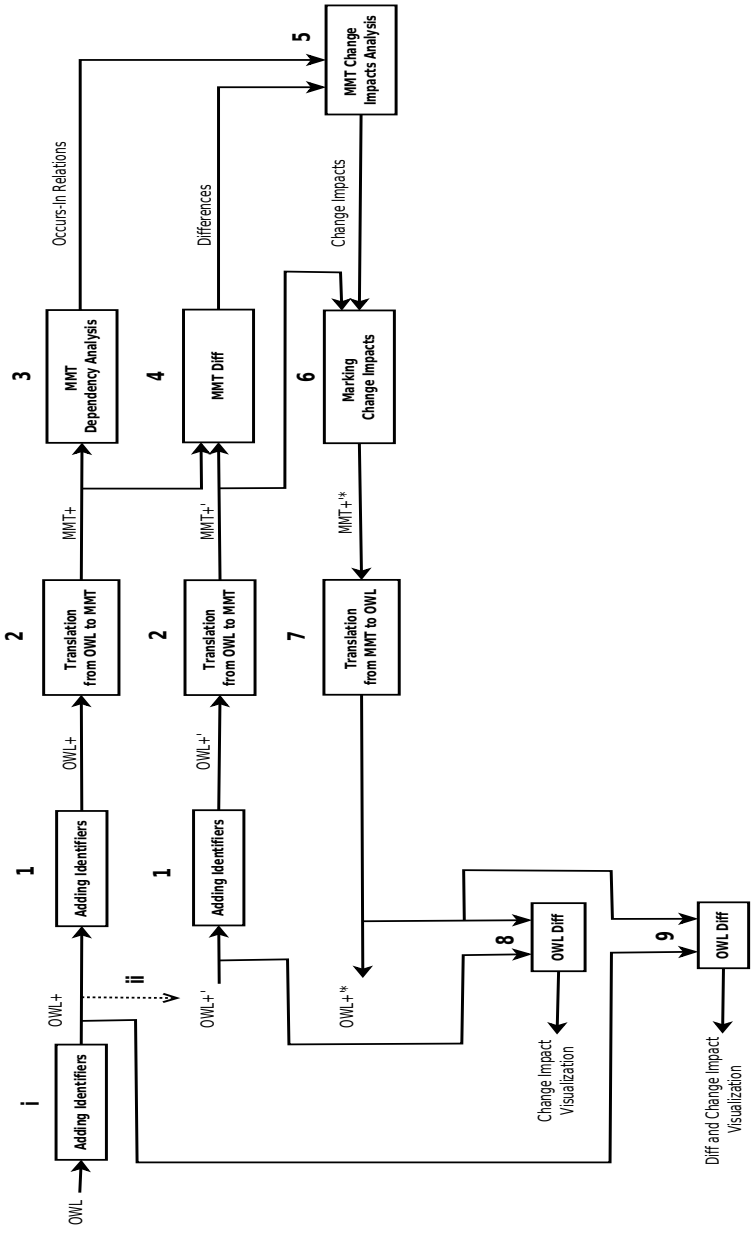


Figure 6.2: The second workflow of management of change in OWL via MMT

identifier invention that requires those two criteria is necessary to obtain better results from MMT Diff.

We need to inform OWL users of change impacts since they do not work with MMT documents. Representing change impacts to them requires to know which element in OWL a changed element in MMT corresponds to. Therefore, we need to refer to changed elements in order to represent their impacts on elements in an ontology. In other words, we need to refer to changed entities and changed axioms that have effects on axioms and entities in the ontology, respectively.

Axioms in OWL ontologies correspond to constant declarations in MMT theories. Constant declarations are identified by constant names. For this reason, we add identifiers to axioms and use the identifiers of the axioms as identifiers of the corresponding constants. An entity declaration axiom consists of an IRI which identifies the entity. We use the IRI as an identifier of the corresponding constant by setting the constant name to the IRI. Therefore, we only need to add identifiers to axioms other than entity declaration axioms.

We initially ask OWL users to add identifiers, in the predefined structure, to axioms in both versions of an ontology. In addition to this, our system automatically adds identifiers to axioms. The former satisfies both of the requirements, and the latter satisfies only the first requirement.

As a first solution approach, we could give numbers to axioms. This approach would satisfy the first condition, which means every MMT constant has a unique name, but it would not satisfy the second requirement since axioms are translated to constants in random order via the translation system.

As a second solution approach, we could just give the same name to all axioms, thus satisfying the second requirement. However, this approach would fail to assign a unique name for each axiom. Therefore, we need a solution that satisfies both of the two requirements.

The Scala Method *hashCode* We decided to use the Scala method *hashCode* to assign a unique value for each axiom with no identifier. Our implementation calls the *hashCode* method for each axiom without identifiers in order to assign a unique hash code value for each axiom as an identifier. In this way, the same axioms in versions of an ontology in OWL are assigned to the same hash code value by the *hashCode* method. In terms of MMT, our implementation translates each axiom to a constant, and assigns the identifier value i.e. hashCode value of the axiom to the name of the constant. Thus, the same constants in versions of theories in MMT are set to the same hash code value.

We observed how the *hashCode* method assigns values to axiom identifiers

and constant names in practice by changing some axioms in the extended family ontology and then adding identifiers via our implementation.

First, we changed axioms in the ontology in different ways, and then added identifiers to axioms in both versions by our implementation. An identifier with a hash code value was added to each axiom that has no identifier. We observed that axioms preserved their identifiers for each of the following cases.

- Changing the order of the object property assertion axiom stating that Liz's husband is Max and a class assertion axiom stating that Max is a Father,
- deleting the object property assertion axiom for Liz,
- adding the object property assertion axiom for Liz back to the ontology,
- changing the equivalent class axiom stating that a parent is either a mother or a father to an axiom stating that a mother or a father is a parent,
- duplicating the class assertion axiom for Max. Note that, in this case, one of the two axioms was dropped and the other one gets an identifier.

Second, we translated both versions of the ontology to MMT, and then for each axiom, our implementation set the name of the corresponding constant to a hash code value. Axioms without identifiers were translated to constants with unique constant names generated by the *hashCode* method. We observed that the names of the constants that correspond to the axioms did not change for translating them to MMT again as well as for each of the change case above.

However, when we updated an axiom, its identifier and the name of the corresponding constant were set to different hashCode values in two versions. For example, we changed the object property assertion axiom stating that Liz's brother is Tom to another object property assertion axiom stating that Liza's brother is Tom. Then, the identifier of the axiom was changed from ax464831953, shown in Figure 6.3, to ax34856183, shown in Figure 6.4, and similarly, the name of the corresponding constant was changed from the old value, shown in Figure 6.5, to the new value, shown in Figure 6.6.

Note that MMT examples that we use in this thesis have base values
`http://cds.omdoc.org/logics/description/owl/owl.omdoc`
`http://cds.omdoc.org/logics/description/owl/owl2.omdoc`
`http://example.com/owl/families`.
We will denote them with A, B, C, respectively.

```
<ObjectPropertyAssertion>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/
       identifier#id"/>
  <Literal datatypeIRI="xsd:string">
       ax464831953</Literal>
 </Annotation>
 <ObjectProperty IRI="hasBrother"/>
 <NamedIndividual IRI="Liz"/>
 <NamedIndividual IRI="Tom"/>
</ObjectPropertyAssertion>
```

Figure 6.3: An example of an axiom identifier produced by the hashCode method

```
<ObjectPropertyAssertion>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/
       identifier#id"/>
  <Literal datatypeIRI="xsd:string">
       ax34856183</Literal>
 </Annotation>
 <ObjectProperty IRI="hasBrother"/>
 <NamedIndividual IRI="Liza"/>
 <NamedIndividual IRI="Tom"/>
</ObjectPropertyAssertion>
```

Figure 6.4: An example of an updated axiom identifier produced by the hashCode method

```
<constant name="ax464831953">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMA>
   <om:OMS module="OWL2SUB"
    name="objectPropertyAssertion" base="B"></om:OMS>
    <om:OMS module="_"
     name="hasBrother" base="C"></om:OMS>
    <om:OMS module="_"
     name="Liz" base="C"></om:OMS>
    <om:OMS module="_"
     name="Tom" base="C"></om:OMS>
  </om:OMA>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.5: An example of a constant name produced by the hasCode method

We offer two options to OWL users for adding identifiers to axioms, which yields two types of identifiers: user-added and auto-added identifiers.

First, we inform users that they are required to add identifiers to axioms in ontologies if they need to obtain more precise results. They add identifiers to axioms as annotations with the predefined structure according to their needs. That means, they may add identifiers to all, some or none of the axioms. Second, our implementation adds identifiers to axioms in OWL documents, and then the users are expected to continue working on these documents. In other words, we get OWL documents from users and automatically add identifiers to axioms. We annotate only axioms which do not have an identifier, and then we save the ontologies with identifiers as separate OWL files. Users work on the new OWL documents and change the ontologies. After that, we translate those ontologies to

```
<constant name="ax34856183">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMA>
   <om:OMS module="OWL2SUB"
    name="objectPropertyAssertion" base="B"></om:OMS>
    <om:OMS module="_"
     name="hasBrother" base="C"></om:OMS>
    <om:OMS module="_"
     name="Liza" base="C"></om:OMS>
    <om:OMS module="_"
     name="Tom" base="C"></om:OMS>
  </om:OMA>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.6: An example of an updated constant name produced by the hasCode method

their representations in MMT, and set the corresponding constant names to the identifier values.

6.4 Applying MMT Diff

We apply the MMT Diff to two versions of an OWL ontology that are translated to MMT in order to find out differences between them. MMT Diff detects and lists changes between the two versions as deletion, addition, renaming and updating.

Changes of entity declarations and axioms are listed as changes of the corresponding constants by MMT Diff, because entity declarations and axioms correspond to constants of type symbol and application, respectively in MMT. We explain outputs of MMT Diff for each kind of changes between two versions of an ontology translated to MMT in the following sections.

6.4.1 Renaming

If a constant of type symbol has been renamed, MMT Diff detects the change as a renaming. For example,a class `Person` has been renamed to `Human`, shown in Figure 6.7, in the example ontology. In other words, its representation in MMT, which corresponds to a constant with name `Person` has been renamed to `Human`, shown in Figure 6.8.

MMT Diff detects this change as a renaming and lists the name of the current constant `Human` and the path of the previous constant with name `Person` as shown in Figure 6.9.

```
<Declaration>
 <Class IRI="Person"/>
</Declaration>

<Declaration>
 <Class IRI="Human"/>
</Declaration>
```

Figure 6.7: An example of a renamed entity

```
<constant name="Person">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMS module="OWLBase"
   name="class" base="A">
  </om:OMS>
 </om:OMOBJ>
 </type>
</constant>

<constant name="Human">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMS module="OWLBase"
   name="class" base="A">
  </om:OMS>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.8: An example of a renamed constant of type symbol

```
<omdoc-diff>
 <declaration name="Human" type="Constant" path="http://example.com/owl/families?_?Person" change=
      "rename">
 </declaration>
</omdoc-diff>
```

Figure 6.9: An example of detecting and listing an renamed constant of type symbol by MMT Diff

6.4.2 Deletion

If a constant of type symbol has been deleted, MMT Diff detects the change as a deletion. For example, a class `ChildlessPerson`, shown in Figure 6.10, has been deleted in the example ontology. In other words, its representation in MMT, which corresponds to a constant with name `ChildlessPerson`, shown in Figure 6.11, has been deleted.

```
<Declaration>
 <Class IRI="ChildlessPerson"/>
</Declaration>
```

Figure 6.10: An example of a deleted entity

```
<constant name="ChildlessPerson" >
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMS module="OWLBase" name="class" base="A"></om:OMS>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.11: An example of a deleted constant of type entity

MMT Diff detects this change as a deletion and prints the path of the constant with name `ChildlessPerson` as shown in Figure 6.12.

```
<omdoc-diff>
 <declaration path="http://example.com/owl/families?_?ChildlessPerson" change="delete">
 </declaration>
</omdoc-diff>
```

Figure 6.12: An example of detecting and listing a deleted constant of type symbol by MMT Diff

Similarly, if a constant of type application has been deleted, MMT Diff detects the change as a deletion. For instance, an assertion axiom with a user-added identifier `ID002` stating that Alice is Max's daughter, shown in Figure 6.13, has been removed from the example ontology. In terms of MMT, a constant with name `ID002`, shown in Figure 6.14, has been deleted.

MMT Diff detects this change as a deletion and prints the path of the constant with name `ID002` as shown in Figure 6.15.

```
<ObjectPropertyAssertion>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
  <Literal datatypeIRI="xsd:string">ID002</Literal>
 </Annotation>
 <ObjectProperty IRI="hasDaughter"/>
 <NamedIndividual IRI="Max"/>
 <NamedIndividual IRI="Alice"/>
</ObjectPropertyAssertion>
```

Figure 6.13: An example of a deleted axiom

```
<constant name="ID002">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMA>
   <om:OMS module="OWL2SUB" name="objectPropertyAssertion" base="B"></om:OMS>
    <om:OMS module="_" name="hasDaughter" base="C"></om:OMS>
    <om:OMS module="_" name="Max" base="C"></om:OMS>
    <om:OMS module="_" name="Alice" base="C"></om:OMS>
  </om:OMA>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.14: An example of a deleted constant of type application

```
<omdoc-diff>
 <declaration  path="http://example.com/owl/families?_?ID002" change="delete">
 </declaration>
</omdoc-diff>
```

Figure 6.15: An example of detecting and listing a deleted constant of type application by MMT Diff

6.4.3 Addition

If a constant of type symbol has been added, MMT Diff detects the change as an addition. For example, an individual `Bob`, shown in Figure 6.16, has been added to the example ontology. In other words, its representation in MMT, which corresponds to a constant with name `Bob`, shown in Figure 6.17, has been added.

```
<Declaration>
 <NamedIndividual IRI="Bob"/>
</Declaration>
```

Figure 6.16: An example of an added entity

```
<constant name="Bob">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMS module="OWLBase" name="individual" base="A"></om:OMS>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.17: An example of an added constant of type symbol

MMT Diff detects this change as an addition and lists the path of the constant with name `Bob` as shown in Figure 6.18.

Similarly, if a constant of type application has been added, MMT Diff detects the change as an addition. For instance, an assertion axiom stating that Bob is Amy's boyfriend, shown in Figure 6.19, has been added to the example ontology. In terms of MMT, a constant with name `ax600113139`, shown in Figure 6.20, has been added.

MMT Diff detects this change as an addition and lists the path of the constant with name `ax600113139` as shown in Figure 6.21.

```
<omdoc-diff>
 <declaration path="http://example.com/owl/families?_?Bob" change="add">
 </declaration>
</omdoc-diff>
```

Figure 6.18: An example of detecting and listing an added constant of type symbol by MMT Diff

```
<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasBoyfriend"/>
 <NamedIndividual IRI="Amy"/>
 <NamedIndividual IRI="Bob"/>
</ObjectPropertyAssertion>
```

Figure 6.19: An example of an added axiom

```
<constant name="ax600113139">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMA>
   <om:OMS module="OWL2SUB" name="objectPropertyAssertion" base="B"></om:OMS>
    <om:OMS module="_" name="hasBoyfriend" base="C"></om:OMS>
    <om:OMS module="_" name="Amy" base="C"></om:OMS>
    <om:OMS module="_" name="Bob" base="C"></om:OMS>
  </om:OMA>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.20: An example of an added constant of type application

```
<omdoc-diff>
 <declaration  path="http://example.com/owl/families?_?ax600113139" change="add">
 </declaration>
</omdoc-diff>
```

Figure 6.21: An example of detecting and listing an added constant of type application

6.4.4 Updating

MMT Diff compares two constants with the same name, which correspond to two axioms with the same identifiers in OWL. If there has been any modification on the constants with the same name, it marks that change as an update. Since we can assign the two kinds of identifiers to axioms, which are user-added and auto-added identifiers, we give an example for each kind of identifiers.

First example, an assertion axiom with a user-added identifier `ID001` stating that Max's last name is Evans, shown in Figure 6.22, has been updated to Max's surname is Evans, shown in Figure 6.23, because of renaming an object property `hasLastname` to `hasSurname`. In other words, its representation in MMT, which is a constant with name `ID001`, shown in Figure 6.24, has been updated to a constant with the same name, shown in Figure 6.25.

```
<DataPropertyAssertion>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
  <Literal datatypeIRI="xsd:string">ID001</Literal>
 </Annotation>
 <DataProperty IRI="hasLastName"/>
 <NamedIndividual IRI="Max"/>
 <Literal datatypeIRI="xsd:string">Evans</Literal>
</DataPropertyAssertion>
```

Figure 6.22: An example of an axiom with a user-added identifier

```
<DataPropertyAssertion>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
  <Literal datatypeIRI="xsd:string">ID001</Literal>
 </Annotation>
 <DataProperty IRI="hasSurname"/>
 <NamedIndividual IRI="Max"/>
 <Literal datatypeIRI="xsd:string">Evans</Literal>
</DataPropertyAssertion>
```

Figure 6.23: An example of an updated axiom with a user-added identifier

MMT Diff can compare these two constants and figure out that the change is an update, and then explicitly outputs that the change on the constant is an update and what the content of the update is, shown Figure 6.26.

Second example, an axiom with an auto-added identifier `ax1241281692` stating that every woman is a person, shown in Figure 6.27 has been updated to every woman is a human, shown in Figure 6.28, due to the renaming of the class `Person` to `Human`.

```
<constant name="ID001">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMA>
   <om:OMS module="OWL2SUB" name="dataPropertyAssertion" base="B"></om:OMS>
    <om:OMS module="_" name="hasLastName" base="C"></om:OMS>
    <om:OMS module="_" name="Max" base="C"></om:OMS>
    <om:OMA>
     <om:OMS module="OWL2SUB" name="literal" base="B"></om:OMS>
     <om:OMSTR>Evans</om:OMSTR>
     <om:OMS module="OWL1Datatype" name="string" base="A"></om:OMS>
    </om:OMA>
  </om:OMA>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.24: An example of a constant of type application

```
<constant name="ID001">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMA>
   <om:OMS module="OWL2SUB" name="dataPropertyAssertion" base="B"></om:OMS>
    <om:OMS module="_" name="hasSurname" base="C"></om:OMS>
    <om:OMS module="_" name="Max" base="C"></om:OMS>
    <om:OMA>
     <om:OMS module="OWL2SUB" name="literal" base="B"></om:OMS>
     <om:OMSTR>Evans</om:OMSTR>
     <om:OMS module="OWL1Datatype" name="string" base="A"></om:OMS>
    </om:OMA>
  </om:OMA>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.25: An example of an updated constant of type application

```
<omdoc−diff>
 <component name=”type” path=”http://example.com/owl/families?_?ID001” change=”update”>
 <om:OMA>
  <om:OMS module=”OWL2SUB” name=”dataPropertyAssertion” base=”B”></om:OMS>
   <om:OMS module=”_” name=”hasSurname” base=”C”></om:OMS>
   <om:OMS module=”_” name=”Max” base=”C”></om:OMS>
   <om:OMA>
    <om:OMS module=”OWL2SUB” name=”literal” base=”B”></om:OMS>
    <om:OMSTR>Evans</om:OMSTR>
    <om:OMS module=”OWL1Datatype” name=”string” base=”B”></om:OMS>
   </om:OMA>
 </om:OMA>
 </component>
</omdoc−diff>
```

Figure 6.26: An example of detecting and listing an updated constant of type application with user-added name by MMT Diff

```
<SubClassOf>
 <Annotation>
  <AnnotationProperty IRI=”http://omdoc.org/identifier#id”/>
  <Literal datatypeIRI=”xsd:string”>ax1241281692</Literal>
 </Annotation>
 <Class IRI=”Woman”/>
 <Class IRI=”Person”/>
</SubClassOf>
```

Figure 6.27: An example of an axiom with auto-added identifier

```
<SubClassOf>
 <Annotation>
  <AnnotationProperty IRI=”http://omdoc.org/identifier#id”/>
  <Literal datatypeIRI=”xsd:string”>ax1241281692</Literal>
 </Annotation>
 <Class IRI=”Woman”/>
 <Class IRI=”Human”/>
</SubClassOf>
```

Figure 6.28: An example of an updated axiom with auto-added identifier

```
<constant name="ax1241281692">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMA>
   <om:OMS module="OWL2SUB" name="subClassOf" base="B"></om:OMS>
    <om:OMS module="_" name="Woman" base="C"></om:OMS>
    <om:OMS module="_" name="Person" base="C"></om:OMS>
  </om:OMA>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.29: An example of a constant of type application with auto-added name

```
<constant name="ax1241281692">
 <type>
 <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
  <om:OMA>
   <om:OMS module="OWL2SUB" name="subClassOf" base="B"></om:OMS>
    <om:OMS module="_" name="Woman" base="C"></om:OMS>
    <om:OMS module="_" name="Human" base="C"></om:OMS>
  </om:OMA>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.30: An example of an updated constant of type application with auto-added name

The axiom is represented as a constant with name `ax1241281692`, shown in Figure 6.29 in MMT. That means, the constant has been updated to another constant with the same name, shown in Figure 6.30.

MMT Diff compares these two constants and lists a difference indicating that the change is an update with the content of the current axiom as shown in Figure 6.31.

However, although a constant has been updated, MMT Diff can not detect that change as an update if the name of the older constant and the name of the current constant are not same. Instead, MMT Diff outputs that the change is a deletion and addition operation, which means that the older constant has been deleted and the current constant has been added.

For instance, if the axioms in the previous example do not have identifiers in OWL, the older axiom and its updated version are translated to two constants with different names, shown in Figure 6.32 and Figure 6.33, in MMT.

Therefore, MMT Diff can not compare these two constants. Thus, it detects the change as deleting the older constant and adding the current constant as

```
<omdoc−diff>
 <component name=”type” path=”http://example.com/owl/families?_?ax1241281692”
  change=”update”>
 <om:OMA>
  <om:OMS module=”OWL2SUB” name=”subClassOf” base=”B”></om:OMS>
   <om:OMS module=”_” name=”Woman” base=”C”></om:OMS>
   <om:OMS module=”_” name=”Human” base=”C”></om:OMS>
 </om:OMA>
 </component>
</omdoc−diff>
```

Figure 6.31: An example of detecting and listing an updated constant of type application with auto-added name by MMT Diff

```
<constant name=”ax1949323543”>
 <type>
 <om:OMOBJ xmlns:om=”http://www.openmath.org/OpenMath”>
  <om:OMA>
   <om:OMS module=”OWL2SUB” name=”subClassOf” base=”B”></om:OMS>
    <om:OMS module=”_” name=”Woman” base=”C”></om:OMS>
    <om:OMS module=”_” name=”Person” base=”C”></om:OMS>
  </om:OMA>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.32: An example of a constant with an auto-added identifier

```
<constant name=”ax1408526537”>
 <type>
 <om:OMOBJ xmlns:om=”http://www.openmath.org/OpenMath”>
  <om:OMA>
   <om:OMS module=”OWL2SUB” name=”subClassOf” base=”B”></om:OMS>
   <om:OMS module=”_” name=”Woman” base=”C”></om:OMS>
   <om:OMS module=”_” name=”Human” base=”C”></om:OMS>
  </om:OMA>
 </om:OMOBJ>
 </type>
</constant>
```

Figure 6.33: An example of a constant with a changed auto-added identifier

```
<omdoc−diff>
 <declaration path="http://example.com/owl/families?_?ax1949323543" change="delete">
 </declaration>

<declaration change="add">
 <constant name="ax1408526537" >
  <type>
  <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
   <om:OMA>
    <om:OMS module="OWL2SUB" name="subClassOf" base="B"></om:OMS>
    <om:OMS module="_" name="Woman" base="C"></om:OMS>
    <om:OMS module="_" name="Human" base="C"></om:OMS>
   </om:OMA>
  </om:OMOBJ>
  </type>
 </constant>
</declaration>
</omdoc−diff>
```

Figure 6.34: An example of detecting and listing an updated constant with changed auto-added name as delete and add by MMT Diff

shown in Figure 6.34.

6.5 Marking Change Impacts

Change impacts can be computed and then propagated via MMT change impact analysis and propagator, see Section 2.5. We make use of the MMT change impact analysis to identify change impacts in OWL ontologies represented in MMT, and we implement an extension of the propagator in accordance with what we need to mark change impacts, see Section 4.3. Our implementation determines impacts of changes, generates metadata with respect to the impacts and adds them to impacted constants, which means mark impacted constants by meta data.

Axioms in OWL correspond to constants of different types in MMT, see Section 5.2.1. If an entity declaration (a declaration axiom) is changed, axioms that include the entity are affected by the change. Besides, if an axim other than declaration axioms is changed, the modification may have impacts on entities that occur in the axiom. In terms of MMT, if a constant of type symbol is changed, the change affects constants of type application that contain it. A change on a constant of type application may influence its arguments, i.e., constants that are declared as constants of type symbol. Therefore, we distinguish types of changed constants, and set their impacts respectively in the implementation.

As we explain in section 4.2, change types are deletion, addition, renaming

and updating in OWL. All axioms (entity declaration axioms and other axioms) can be modified by deleting and adding them. In contrast to those, renaming can be applied to only declaration axioms, and updating can be applied to only other axioms. In terms of MMT, if a change is a deletion or an addition, it can be a modification of any constant. If the change is a renaming, it means a constant of type symbol has been changed. In contrast to that, if the change is an update, it means a constant of type application has been modified.

Our implementation for marking change impacts consists of three steps: determining change impacts, generating metadata about the impacts and adding the metadata to impacted constants.

The first step: For each change that is returned from the diff, we identify impacts of the change by a binary relation called RefersTo in MMT. If the change is a constant of type symbol, then we use the inverse of the relation. If the change is a constant of type application, then we use the relation. In terms of OWL, the former means that the changed constant refers to an entity (declaration axiom) and thus, the inverse relation returns impacted constants i.e., axioms that includes the changed entity. The latter means that the changed constant is a logical axiom or an annotation axiom, and thus, impacted constants i.e., entities that occur in the changed axiom are obtained via the relation.

The second step: For each impacted constant, we generate metadata with a special key (meta property) and value (application with string) for each kind of changes that have impacts on the constant. Metadata are represented as annotations in OWL. Meta property and meta value correspond to annotation property and annotation value, respectively. Meta property and annotation property are identifiers of change impacts, and meta value and annotation value are type of string that gives information about the impact.

We set the base part of the identifier as `http://omdoc.org/impact#` for both impacts or possible impacts of a change. In addition, we set the fragment part of the identifier, which represents the type of a change that causes an impact, as `impactByDelete`, and `impactByRename` for impacts, and as `possibleImpactByDelete`, `possibleImpactByAdd` and `possibleImpactByUpdate` for possible impacts. In other words, we set the identifiers for impacts and possible impacts of a change with respect to the change types as shown in Section 4.3.1.

We set metadata value as string that gives information about impacts on constants. We define two kinds of string. The first one gives information about impacts of deleting or renaming of a constant of type symbol, and the second

one informs about possible impacts of deleting, adding or updating a constant of type application. In terms of OWL, an annotation value, which is type of string, informs about effects of deleting or renaming of an entity (declaration axiom) as well as deleting, adding or updating of logical axioms and annotation axioms.

Marking Impacts of a Change We use the first kind of string if the impacted constant is of type application. The string states that a change of a constant of type symbol has an impact on a constant that includes this information as metadata, which is written in terms of OWL.

It consists of four parts: base, type of the change, type of the changed constant, and name of the changed constant.

1. The base part is always set to the same string: "This axiom has been impacted by".

2. The type of the change is set to "deleting", or "renaming" with respect to the change type. For example, if the change is a deletion of a constant, then the second part will be "deleting".

3. The type of the changed constant is set to the symbol name in the changed constant, i.e., type of the changed entity. For example, if the changed constant corresponds to a class in OWL, then the third part will be "class".

4. The name of the changed constant is set to the name of the changed constant, i.e., the name of the changed entity. For example, if the changed constant is an entity named "ChildlessPerson", then this part will be "ChildlessPerson".

For impacts, generated string for metadata value and its correspondence in OWL are shown in Figure 6.35 and Figure 6.36, respectively.

```
<metadata>
 <meta property="http://omdoc.org/impact?_?impactByDelete">
  <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
   <om:OMA>
    <om:OMS module="OWL2SUB" name="literal" base="B"> </om:OMS>
    <om:OMSTR>This axiom has been impacted by deleting class ChildlessPerson</om:OMSTR>
    <om:OMS module="OWL1Datatype" name="string" base="A"></om:OMS>
   </om:OMA>
  </om:OMOBJ>
 </meta>
</metadata>
```

Figure 6.35: An example of marking an impact of a change as metadata in MMT

```
<Annotation>
 <AnnotationProperty IRI="http://omdoc.org/impact#impactByDelete"/>
 <Literal datatypeIRI="xsd:string">
  This axiom has been impacted by deleting class ChildlessPerson</Literal>
</Annotation>
```

Figure 6.36: An example of marking an impact of a change as an annotation in OWL

The following two examples show that a constant of type application is affected by deleting a constant of type symbol in MMT, shown in Figure 6.37, and, which means, an axiom is affected by deleting an entity in OWL, shown in Figure 6.38, by stating that "This axiom is impacted by deleting class ChildlessPerson". The translation of the constant in the first example to OWL looks like the axiom in the second example. The second example also shows that the axiom includes the deleted class *ChildlessPerson*.

Marking Possible Impacts of a Change We use the second kind of string if the impacted constant is of type symbol. The string states that a change of a constant of type application may have an impact on a constant that includes this information as metadata, which is also written in terms of OWL. In other words, it informs about a possible impact of changing an axiom.

It consists of three parts: identifier of the changed constant, base, and type of the change.

1. The identifier of the change constant is set to "An axiom with ID " concatenated with the name of the changed constant of type application, i.e. the identifier of the changed axiom. For example, "An axiom with identifier ax-1711766800".

2. The base part is always set to the same string: "that includes this entity has been".

3. The type of the change is set to "deleted", "added", or "updated" with respect to the change type. For example, if the change is a deletion of a constant of type application, then the third part will be "deleted".

For possible impacts, generated string for metadata value and its correspondence in OWL are shown in Figure 6.39 and in Figure 6.40 .

The following two examples indicate that a constant may be affected by updating another constant with name `ax-1711766800` in MMT, shown in Figure 6.41, and, which means, an entity may be affected by deleting an axiom with identifier

```
<constant name="ax−1039132749">
 <metadata>
  <meta property="http://omdoc.org/impact?_?impactByDelete">
   <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
    <om:OMA>
     <om:OMS module="OWL2SUB" name="literal" base="B"> </om:OMS>
     <om:OMSTR>This axiom has been impacted by deleting class ChildlessPerson</om:OMSTR>
     <om:OMS module="OWL1Datatype" name="string" base="A"></om:OMS>
    </om:OMA>
   </om:OMOBJ>
  </meta>
 <type>
  <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
   <om:OMA>
    <om:OMS module="OWL2SUB" name="equivalentClasses" base="B"></om:OMS>
    <om:OMS module="_" name="ChildlessPerson" base="C"></om:OMS>
    <om:OMA>
     <om:OMS module="OWL2SUB" name="objectIntersectionOf" base="B"></om:OMS>
     <om:OMS module="_" name="Person" base="C"></om:OMS>
     <om:OMA>
      <om:OMS module="OWL2QLRL" name="objectComplementOf" base="B">
      </om:OMS>
      <om:OMS module="_" name="Parent" base="C"></om:OMS>
     </om:OMA>
    </om:OMA>
   </om:OMA>
  </om:OMOBJ>
 </type>
</constant>
```

Figure 6.37: An example of marking an impact of deleting a constant of type symbol

```
<EquivalentClasses>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#impactByDelete"/>
  <Literal datatypeIRI="xsd:string">
   This axiom has been impacted by deleting class ChildlessPerson</Literal>
 </Annotation>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
  <Literal datatypeIRI="xsd:string">ax−1039132749</Literal>
 </Annotation>
 <Class IRI="ChildlessPerson"/>
  <ObjectIntersectionOf>
   <Class IRI="Person"/>
   <ObjectComplementOf>
    <Class IRI="Parent"/>
   </ObjectComplementOf>
  </ObjectIntersectionOf>
</EquivalentClasses>
```

Figure 6.38: An example of marking an impact of deleting an entity

```
<metadata>
 <meta property="http://omdoc.org/impact?_?possibleImpact">
  <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
   <om:OMA>
    <om:OMS module="OWL2SUB" name="literal" base="B"></om:OMS>
    <om:OMSTR> An axiom with ID ax-1711766800 that includes this entity has been deleted.</
        om:OMSTR>
    <om:OMS module="OWL1Datatype" name="string" base="A"></om:OMS>
   </om:OMA>
  </om:OMOBJ>
 </meta>
</metadata>
```

Figure 6.39: An example of marking a possible impact of a change as metadata in MMT

```
<Annotation>
 <AnnotationProperty IRI="http://omdoc.org/impact#possibleImpact"/>
 <Literal datatypeIRI="xsd:string">
  An axiom with ID ax-1711766800 that includes this entity has been deleted.</Literal>
</Annotation>
```

Figure 6.40: An example of marking a possible impact of a change as an annotation in OWL

`ax-1711766800` in OWL, shown in Figure 6.42, by stating that "An axiom with identifier ax-1711766800 including this entity has been updated". The translation of the constant in the first example to OWL is the entity declaration in the second example.

We also give examples for both impacts and possible impacts of other change types addition, updating and renaming at `https://svn.kwarc.info/repos/MMT/src/mmt-owl/ExtendedFamilyOntology`.

The third step: We add information about each change which affects the impacted constant as metadata. We take the generated metadata with respect to the impacts in the second step and add them to the impacted constant. In terms of OWL, we can see information about each change which affects an axiom as an annotation in the axiom. We do not directly generate an annotation with respect to impacts. Instead, we translate generated metadata in constants to annotations in axioms. Generated and added metadata and corresponding annotations for impacts, shown in Figure 6.37 and in Figure 6.38 and for possible impacts, shown in Figure 6.41 and in Figure 6.42 are given in the examples.

```
<constant name="Liz">
 <metadata>
  <meta property="http://omdoc.org/impact?_?possibleImpact">
   <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
    <om:OMA>
     <om:OMS module="OWL2SUB" name="literal" base="B"></om:OMS>
     <om:OMSTR> An axiom with ID ax−1711766800 that includes this entity has been deleted.
     </om:OMSTR>
     <om:OMS module="OWL1Datatype" name="string" base="A"></om:OMS>
    </om:OMA>
   </om:OMOBJ>
  </meta>
 </metadata>
 <type>
  <om:OMOBJ xmlns:om="http://www.openmath.org/OpenMath">
   <om:OMS module="OWLBase" name="individual" base="A"></om:OMS>
  </om:OMOBJ>
 </type>
</constant>
```

Figure 6.41: An example of marking a possible impact of updating a constant of type application

```
<Declaration>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/impact#possibleImpact"/>
  <Literal datatypeIRI="xsd:string">
   An axiom with ID ax−1711766800 that includes this entity has been deleted.</Literal>
 </Annotation>
 <NamedIndividual IRI="Liz"/>
</Declaration>
```

Figure 6.42: An example of marking a possible impact of updating an axiom

6.6 Visualization of Changes and their Impacts

We can use existing OWL diff tools to visualize changes and their impacts between two versions of an ontology if they support detecting annotation changes as well. There are two options to present differences and change impacts.

In the first option, we apply an OWL diff tool to a previous version and the current version with impacts of an ontology to display differences and change impacts together.

In the second one, we apply the tool to the current version and the current version with impacts of an ontology to show only change impacts.

The existing diff tools do not support detecting annotation changes or comparing OWL 2 ontologies. We explain some of the ontology comparison tools such as Ecco, OWLDiff and OWL Diff Tool.

Ecco [GPS12] is a recently implemented hybrid diff tool for OWL 2 ontologies. It distinguishes effectual and ineffectual changes between ontologies and presents them by categorizing. OWLDiff [kri11] is a tool for comparison and merge for OWL ontologies. It represents changes in both ontology versions in form of trees. However, both of these tools do not support annotation changes.

OWL Diff Tool [RN] is another difference tool which is available as a plugin in the latest Protégé editor. It detects and shows differences such as adding, deleting and renaming of entities and annotation changes. However, it does not give a detailed description for annotation changes.

We can use View functionality in the user interface of the Protégé editor to present change impacts. Many of the views are not visible by default, so users must configure the user interface by activating necessary views in **View** menu. For details of the **View** menu and how to configure it, see
`http://protegewiki.stanford.edu/wiki/Protege4GettingStarted` and
`http://protegewiki.stanford.edu/wiki/Protege4Views`.

Change Impacts can be displayed in three ways. We describe each of them in the following paragraphs.

The first way: We use `Annotation properties` tab, and we activate two Annotation property views which are **Annotation property hierarchy** and **Usage** and one Misc view which is **Axioms** by clicking on the **View** menu. The Annotation property hierarchy view lists annotation properties such as *impactByDelete* and *impactByRename* for impact annotations, and *possibleImpactByAdd*, *possibleImpactByDelete* and *possibleImpactByUpdate* for possible impact annotations. The Usage view shows references of the currently selected entity, and expressions in the list are hyperlinked for navigation. The Axioms view lists all axioms with their annotations.

After the configuration, change impacts can be seen as annotations by following 4 steps:

1. we select one of the impact annotation properties in the Annotation property hierarchy view,

2. we see impacted axioms in the Usage view,

3. we find one of the impacted axioms in the Axiom view,

4. we click on the highlighted @ icon on the right side of the axiom to read impact annotations.

For example, we can see change impacts on axioms which are impacted by deleting operations. First, we click on *impactByDelete* in the Annotation property hierarchy view. Second, we see a list of axioms impacted by deleting in the Usage view. Third, we find an axiom stating that Alice is Max's daughter in the Axiom view. Last, we click on @ icon on the right side of the axiom, and thus, we see an impact annotation saying that "This axiom has been impacted by deleting individual Alice" in a pop up window as shown in the following Figure 6.43.

The second way: We use `Annotation properties` tab, and we activate two Annotation property views which are **Annotation property hierarchy** and **Usage** again by clicking on the **View** menu. We do not need the **Axioms** since we use the hyperlinks in the Usage view instead of searching for an axiom in the list in the Axiom view.

Change impacts can be presented as annotations by following 6 steps:

1. we select one of the impact annotation properties in the Annotation property hierarchy view,
2. we see the impacted axioms in the Usage view,
3. we click on the entity part of one of the impacted axioms in the Usage view to move to the Entities tab via hyperlinks,
4. we click on the entity in the Entities tab to see axioms related to it,
5. we see the impacted axioms related to the entity,
6. we click on the highlighted @ icon on the right side of the axiom to read impact annotations.

For instance, we can see change impacts on the same axiom again by the second way. First, we click on *impactByDelete* in the Annotation property hierarchy view. Second, we see a list of axioms impacted by deleting in the Usage view. Third, we click on `Max` which is the entity part of the axiom stating that Alice is Max's daughter in the Usage view as shown in the Figure 6.44. Then, we automatically move to the Entities tab via hyperlink of the entity `Max`. Fourth, we click on the entity `Max` and see all related axioms to it. Fifth, we see the property assertion of the `Max` which is `hasDaughter Alice`. Finally, we click on the highlighted @ icon on the right side of the axiom to read impact annotation saying that "This axiom has been impacted by deleting individual Alice" in a pop up window as shown in the following Figure 6.45.

The third way: Similarly, we use `Annotation properties` tab, and we activate two Annotation property views which are **Annotation property hierarchy** and **Usage** again by clicking on the **View** menu. This time we need neither the **Axioms** nor hyperlinks in the Usage view since we use explanation lines in the Usage view.

Change impacts can be displayed as annotations by following 4 steps:

1. we select one of the impact annotation properties in the Annotation property hierarchy view,
2. we see the impacted axioms in the Usage view,
3. we move the mouse pointer to one of the axioms in the Usage view,
4. we see a line of explanation about annotations of the axiom.

For example, we can see a line of explanation about change impacts on the same axiom by the third way as in the following Figure 6.46.

Figure 6.43: First way of visualizing change impacts

Figure 6.44: Second way of visualizing change impacts - step 1

Figure 6.45: Second way of visualizing change impacts - step 2

Figure 6.46: Third way of visualizing change impacts

Chapter 7

Conclusion

7.1 Summary

In this thesis, we have investigated whether generic MMT services can be applied to specific domains, specifically, applied the generic MMT management of change service to OWL, and thus provided a management of change service for OWL ontologies. We have implemented a system to apply the MMT MoC service to OWL. It covers detecting changes, identifying impacts of the changes, and marking impacted elements.

Firstly, we assessed requirements for change impact analysis in OWL ontologies. The requirements are determining dependency relations between elements in ontologies, identifying and representing change impacts.

We determined dependency relations between entities, expressions and axioms in OWL ontologies. Also, we considered deletion, addition, renaming and updating as the types of changes that can be applied to OWL ontologies. Note that any other kind of change can be expressed as a combination of these changes. Based on the dependency relations, we identified change impacts. In particular, we applied the four kinds of changes separately and all together to the extended family ontology and identified impacts and possible impacts of the changes. In addition, we established requirements of representing changes and their impacts, and developed a solution that satisfies the requirements, which is the representation of changes and their impacts as annotations in ontologies. One of the essential requirements was to distinguish changes and their impacts from other annotations. Therefore, we decided to define a special annotation structure with an annotation property for impacts and possible impacts of each kind of changes, which allowed us to distinguish not only annotations about changes and their impacts from other annotations but also different impacts from each other.

We developed a system to apply the management of change service for MMT to OWL ontologies. We need to add identifiers to axioms to get more precise results from the system as well as to refer to changed elements in OWL. Our system takes two versions of an ontology, adds identifiers to axioms in both ontology versions, and returns changes between them and impacts of the changes on the current ontology version. Changes are identified as deletion, addition, renaming, and updating operations. In our second approach, the system adds identifiers to axioms in the original ontology, and then OWL users are required to apply changes to that derived ontology with identifiers from the ontology in order to improve the results of our system.

The work flows of both approaches are the same after adding identifiers to axioms in ontologies, which are translating OWL ontologies to their representations in MMT, determining dependency relations between elements in the previous ontology version, applying MMT Diff to both the previous and the current ontology versions to detect changes, making use of MMT CIA to identify impacts of the changes, adding change impact annotations to the current ontology version in MMT, and translating the MMT document with impact annotations to its representation in OWL, and finally, visualizing changes and their impacts via OWL editors or diff tools.

Users can visualize change impacts by OWL editors or diff tools. An OWL diff tool can be applied to a previous version of an ontology with identifiers and the current version of the ontology with impact annotations to display changes and their impacts together if the diff tool supports detecting changes of annotations. In addition, the tool can be applied to the current ontology version with identifiers and the current ontology version with impact annotations to show only change impacts.

Alternatively, change impacts can be displayed by OWL editors. We used the latest Protégé editor to realize the visualization by the view functionality of the editor. Since many of the views are not visible by default, we explained how to configure the user interface to include necessary views. Additionally, we found out three methods of the visualization, and gave instructions on how to display change impact annotations by each method.

7.2 Discussion

We have evaluated the work described in this thesis by using the various versions of the extended family ontology. The management of change service for OWL ontologies via MMT was based on the bi-directional translation between OWL and

MMT, thus gave us the advantage of developing services for MMT and applying them to OWL ontologies.

Our discussion on requirements for change management in OWL ontologies covers only entities, expressions and axioms in OWL. Dependency relations between those elements were determined systematically. In light of these sets of knowledge, applying the four kinds of changes to the ontology separately and all together enabled us to discuss impacts of each change type separately and impacts of all change types together, respectively. The consideration of changes and their impacts in detail contributed to determining what to expect from an automated change impact analysis for OWL. However, we could not observe all possible impacts of the changes since the ontologies do not include all possible combinations of the elements.

Furthermore, our discussion on requirements of representing change impacts was beneficial for producing the solution for the representation, which was to use annotations in ontologies. The special annotation structure that we defined for impacts of changes permitted us to distinguish change impact annotations from other annotations. Also, the predefined annotation properties for impacts and possible impacts of each kind of changes allowed us to separate different impacts from each other.

Moreover, the entire discussion enabled us to determine what any management of change system for OWL ontologies needs to support and how we should implement the system for OWL ontologies.

The system that we developed to apply the management of change service for MMT to OWL ontologies worked more precisely when axioms had identifiers added by users. Otherwise, updating operations were identified as delete and add operations. Our second approach improved the results of our system. In this way, we obtained more precise results such as detecting a change as an update.

Our implementation makes visualization of change impacts possible by OWL editors or diff tools since we use annotations for the representation of change impacts. Annotations are readily available in OWL and they should be supported by any OWL tool. OWL diff tools can display changes and their impacts together or only change impacts. However, we applied the existing OWL diff tools to versions of the extended family ontology, and we could not see change impact annotations since the tools do not support detecting annotation changes.

Additionally, change impact annotations can be displayed by OWL editors. There are three methods to present changes impact annotations by the latest Protégé editor. We opened the versions of the extended family ontology with the editor, and observed that the three methods successfully displayed the change impact annotations. The methods have both advantages and disadvantages. In

the first method, even though the users do not have to take many steps to see the impact on an affected element, they have to take a step to find the element in a list of axioms. Finding the element may take time or may be hard for the users if the axioms list is too long, i.e. the ontology contains too many axioms. In the second method, the users have to take more steps to see the annotation via hyperlinks, nevertheless, they do not have to look for the element in the list of axioms. In the third method, the users do not have to use the axiom list or the hyperlinks. Instead, they can use explanation lines. That means, they can see the annotation in an explanation line that appears on the element when the mouse pointer is moved to it. However, it may be hard to read the annotation in the one thin explanation line. The usability evaluation should be done systematically with real ontologies and test users.

From an OWL perspective, OWL users can take advantage of our automated change management service for OWL ontologies since identifying and tracing impacts of changes is not only crucial but also difficult in large ontologies. Moreover, OWL users can benefit from viewing change impacts by OWL editor or diff tools. In addition, they are already familiar with those tools, so they do not have to work with other tools. Similarly, they do not have to know MMT since change impacts are represented as annotations in an OWL ontology itself.

From an MMT perspective, our work was one experiment in order to evaluate whether generic MMT services can be applied to specific domains. We achieved providing a management of change service for OWL ontologies via the generic MMT management of change service. Thus, we demonstrated that generic MMT services can be applied to OWL ontologies.

7.3 Future Work

Further developments can include extending the followings:

1. the management of change service to support change impacts of imports in OWL,

2. the management of change service to support applying changes to ontologies in OWL,

3. OWL diff tools to support changes of annotations.

We detail them in the following paragraphs.

Change Impacts of Imports An OWL ontology can directly import ontologies to gain access to their elements. Since some elements of the imported ontologies are used in the ontology, a change in one of the imported ontologies may lead to impacts on elements of the ontology that imports it. Therefore, we need to extend our system by supporting change impact annotations arising from imports. In other words, a theory can include another theory, and a change in an included theory may affect other constants in the theory in MMT. Thus, we are required to make our system available to provide metadata about impacts of changes in included theories.

Initially, types of imports, and dependency relations between them and the other elements of OWL ontologies should be discussed respectively as we explained the dependency relations between the other elements in OWL. Based on the dependency relations, effects of changes in imported ontologies on the importing ontology should be identified for each kind of changes. After that, our approach of detecting changes and their impacts could be applied to the previous version and the current version of the imported ontology. In other words, for each change in the included theory, i.e. the imported ontology, impacts of the change on the including theory, i.e. on the importing ontology are identified. Finally, change impact metadata about includes are added to impacted constants, i.e. annotations about the impacts of changes to imported ontologies are added to impacted entity declarations or axioms in the ontology.

Applying Changes to Ontologies In large ontologies, it is not always convenient to send the current version of the ontologies between users. Instead, other users can apply changes to the original ontology to obtain the current version of the ontology. In order to support that, our system can be extended by making use of the change propagation feature of the MMT CIA system.

Changes of Annotations As we explained before, any OWL diff tool can visualize change impact annotations if it supports detecting changes of annotations. It also provides the advantage of displaying change impact annotations all together. In addition, viewing the annotations in Protégé is not sufficient for users working with other OWL editors. Therefore, another development can be extending the existing diff tools.

Seperately from the future investigations above, other generic MMT services can be applied to OWL. For example, the MMT search engine can be applied to OWL ontologies.

Bibliography

[ADD+11] Serge Autexier, Catalin David, Dominik Dietrich, Michael Kohlhase, and Vyacheslav Zholudev. Workflows for the Management of Change in Science, Technologies, Engineering and Mathematics. *CoRR*, abs/1105.2392, 2011.

[BBL05] Franz Baader, Sebastian Brand, and Carsten Lutz. Pushing the EL Envelope. In *In Proc. of IJCAI 2005*, pages 364–369. Morgan-Kaufmann Publishers, 2005.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, USA, 2003.

[CIM10] David Carlisle, Patrick Ion, and Robert Miner. Mathematical Markup Language (MathML), 2010. See `http://www.w3.org/TR/MathML3/`.

[CIM11] David Carlisle, Patrick Ion, and Robert Miner. Extensible Markup Language (XML), 2011. See `http://www.w3.org/XML/`.

[CLLR07] Diego Calvanese, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. 2007.

[DBL01] OilEd: A Reason-able Ontology Editor for the Semantic Web. pages 396–408, 2001.

[DS04] Mike Dean and Guus Schreiber. OWL Web Ontology Language Reference, 2004. See `http://www.w3.org/TR/2004/REC-owl-ref-20040210/`.

[FaC11] FaCT++, 2011. See `http://code.google.com/p/factplusplus/`.

[GPS12] Rafael S. Gonçalves, Bijan Parsia, and Ulrike Sattler. Ecco: A Hybrid Diff Tool for OWL 2 Ontologies. In *OWLED*, 2012.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[HKP+09] Pascal Hitzler, Markus Krtzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language Primer, 2009. See `http://www.w3.org/TR/owl2-primer`.

[Hug06] Baden Hughes. Change Management and Versioning in Ontologies, 2006. see `http://www.slideshare.net/badenhughes/change-management-and-versioning-in-ontologies`.

[IR12] M. Iancu and F. Rabe. Management of Change in Declarative Languages. In J. Campbell, J. Carette, G. Dos Reis, J. Jeuring, P. Sojka, V. Sorge, and M. Wenzel, editors, *Intelligent Computer Mathematics*, volume 7362 of *Lecture Notes in Computer Science*, pages 325–340. Springer, 2012.

[Koh06] Michael Kohlhase. OMDoc – *An Open Markup format for Mathematical Documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, 2006.

[KPS+05] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. SWOOP: A Web Ontology Editing Browser. *Journal of Web Semantics*, 4:2005, 2005.

[kri11] OWL Diff, 2011. See `http://krizik.felk.cvut.cz/km/owldiff/index.html/`.

[LGP11] GNU Lesser General Public License, 2011. See `http://www.gnu.org/licenses/lgpl-2.1.html/`.

[MÏ0] N. Müller. Change Management on Semi-Structured Documents, 2010.

[Mad92] Nazim H. Madhavji. Environment Evolution: The Prism Model of Changes, 1992. See `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=135771`.

[MGH+09] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles, 2009. See http://www.w3.org/TR/owl2-profiles/.

[MHSV04] Eetu Makela, Eero Hyvonen, Samppa Saarela, and Kim Viljanen. OntoViews - A Tool for Creating Semantic Web Portals. 2004.

[MMT] The MMT API. https://svn.kwarc.info/repos/MMT/deploy/apidocs/index.html.

[MPSG09] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau. OWL 2 Web Ontology Language: Direct Semantics, 2009. See http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/.

[MPSP09] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax, 2009. See http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/.

[MRMJ12] M.Iancu, F. Rabe, M.Kohlhase, and J.Urban. The Mizar Mathematical Library in OMDoc: Translation and Applications. see http://kwarc.info/frabe/Research/IKRU_mizar_11.pdf, 2012.

[NeO11] NeON Toolkit, 2011. See http://neon-toolkit.org/wiki/Main_Page.

[Ope09] Openmath, 2009. http://www.openmath.org.

[Pro11] Protégé, 2011. See http://protege.stanford.edu/.

[PS99] F. Pfenning and C. Schürmann. System Description: Twelf - A Meta-Logical Framework for Deductive Systems. *Lecture Notes in Computer Science*, 1632:202–206, 1999.

[RK12] F. Rabe and M. Kohlhase. A Scalable Module System, 2012. under review, see http://kwarc.info/frabe/Research/mmt.pdf.

[RN] T. Redmond and N. Noy. Computing the changes between ontologies.

[RS09a] F. Rabe and C. Schürmann. A Practical Module System for LF. In J. Cheney and A. Felty, editors, *Proceedings of the Workshop on Logical Frameworks: Meta-Theory and Practice (LFMTP)*, volume LFMTP'09 of *ACM International Conference Proceeding Series*, pages 40–48. ACM Press, 2009.

[RS09b] F. Rabe and C. Schürmann. A Practical Module System for LF. see http://kwarc.info/frabe/Research/lf.pdf, 2009.

[Sch09] Michael Schneider. OWL 2 Web Ontology Language: RDF-Based Semantics, 2009. See http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/.

[UNS11] The United Nations Standard Products and Services Code, 2011. See http://unspsc.org/.

Appendix A

An Extended Family Ontology

```
<?xml version="1.0"?>
<Ontology
        xmlns="http://www.w3.org/2002/07/owl#"
        xml:base="http://example.com/owl/families/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
        ontologyIRI="http://example.com/owl/families/">

        <Declaration>
         <NamedIndividual IRI="John"/>
        </Declaration>

        <Declaration>
         <NamedIndividual IRI="Mary"/>
        </Declaration>

        <Declaration>
         <NamedIndividual IRI="Liz"/>
        </Declaration>

        <Declaration>
         <NamedIndividual IRI="Liza"/>
        </Declaration>

        <Declaration>
         <NamedIndividual IRI="Tom"/>
        </Declaration>

        <Declaration>
         <NamedIndividual IRI="Amy"/>
        </Declaration>

        <Declaration>
         <NamedIndividual IRI="Max"/>
        </Declaration>

        <Declaration>
         <NamedIndividual IRI="Alice"/>
        </Declaration>
```

```
<Declaration>
 <Class IRI="Person"/>
</Declaration>

<Declaration>
 <Class IRI="Man"/>
</Declaration>

<Declaration>
 <Class IRI="Woman"/>
</Declaration>

<Declaration>
 <Class IRI="Father"/>
</Declaration>

<Declaration>
 <Class IRI="Mother"/>
</Declaration>

<Declaration>
 <Class IRI="Parent"/>
</Declaration>

<Declaration>
 <Class IRI="Teenager"/>
</Declaration>

<Declaration>
 <Class IRI="ChildlessPerson"/>
</Declaration>

<Declaration>
 <Class IRI="NarcisticPerson"/>
</Declaration>

<Declaration>
 <Class IRI="JohnsChildren"/>
</Declaration>

<Declaration>
<ObjectProperty IRI="hasAncestor"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="hasParent"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="parentOf"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="hasChild"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="hasFemaleChild"/>
</Declaration>
```

```
<Declaration>
<ObjectProperty IRI="hasRelative"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="hasWife"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="hasHusband"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="hasDaughter"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="hasSon"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="hasBrother"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="hasSister"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="hasSibling"/>
</Declaration>

<Declaration>
 <ObjectProperty IRI="loves"/>
</Declaration>

<Declaration>
 <DataProperty IRI="hasID"/>
</Declaration>

<Declaration>
 <DataProperty IRI="hasName"/>
</Declaration>

<Declaration>
 <DataProperty IRI="hasFamilyName"/>
</Declaration>

<Declaration>
 <DataProperty IRI="hasAge"/>
</Declaration>

<Declaration>
 <DataProperty IRI="age"/>
</Declaration>

<Declaration>
 <Datatype IRI="personAge"/>
```

```
</Declaration>

<Declaration>
 <Datatype IRI="minorAge"/>
</Declaration>

<Declaration>
 <Datatype IRI="majorAge"/>
</Declaration>

<Declaration>
 <Datatype IRI="toddlerAge"/>
</Declaration>

<DatatypeDefinition>
 <Datatype IRI="minorAge"/>
 <DataIntersectionOf>
  <Datatype IRI="personAge"/>
  <DataComplementOf>
   <Datatype IRI="majorAge"/>
  </DataComplementOf>
 </DataIntersectionOf>
</DatatypeDefinition>

<DatatypeDefinition>
 <Datatype IRI="majorAge"/>
 <DatatypeRestriction>
  <Datatype IRI="xsd:integer"/>
  <FacetRestriction facet="xsd:minInclusive">
   <Literal datatypeIRI="xsd:integer">19</Literal>
  </FacetRestriction>
  <FacetRestriction facet="xsd:maxInclusive">
   <Literal datatypeIRI="xsd:integer">150</Literal>
  </FacetRestriction>
 </DatatypeRestriction>
</DatatypeDefinition>

<DatatypeDefinition>
 <Datatype IRI="personAge"/>
 <DatatypeRestriction>
  <Datatype IRI="xsd:integer"/>
  <FacetRestriction facet="xsd:minInclusive">
   <Literal datatypeIRI="xsd:integer">0</Literal>
  </FacetRestriction>
  <FacetRestriction facet="xsd:maxInclusive">
   <Literal datatypeIRI="xsd:integer">150</Literal>
  </FacetRestriction>
 </DatatypeRestriction>
</DatatypeDefinition>

<DatatypeDefinition>
 <Datatype IRI="personAge"/>
 <DataUnionOf>
  <Datatype IRI="majorAge"/>
  <Datatype IRI="minorAge"/>
 </DataUnionOf>
</DatatypeDefinition>

<DatatypeDefinition>
```

```
 <Datatype IRI="toddlerAge"/>
 <DataOneOf>
  <Literal datatypeIRI="xsd:integer">1</Literal>
  <Literal datatypeIRI="xsd:integer">2</Literal>
 </DataOneOf>
</DatatypeDefinition>

<SubClassOf>
 <Class IRI="Woman"/>
 <Class IRI="Person"/>
</SubClassOf>

<SubClassOf>
 <Class IRI="Man"/>
 <Class IRI="Person"/>
</SubClassOf>

<DisjointClasses>
 <Class IRI="Woman"/>
 <Class IRI="Man"/>
</DisjointClasses>

<DisjointUnion>
 <Class IRI="Person"/>
 <Class IRI="Woman"/>
 <Class IRI="Man"/>
</DisjointUnion>

<EquivalentClasses>
 <Class IRI="Person"/>
 <DataAllValuesFrom>
  <DataProperty IRI="hasID"/>
  <Datatype IRI="xsd:nonNegativeInteger"/>
 </DataAllValuesFrom>
</EquivalentClasses>

<SubClassOf>
 <Class IRI="Teenager"/>
 <DataSomeValuesFrom>
  <DataProperty IRI="hasAge"/>
  <DatatypeRestriction>
   <Datatype IRI="xsd:integer"/>
   <FacetRestriction facet="xsd:minExclusive">
    <Literal datatypeIRI="xsd:integer">12</Literal>
   </FacetRestriction>
   <FacetRestriction facet="xsd:maxInclusive">
    <Literal datatypeIRI="xsd:integer">18</Literal>
   </FacetRestriction>
  </DatatypeRestriction>
 </DataSomeValuesFrom>
</SubClassOf>

<EquivalentClasses>
 <Class IRI="ChildlessPerson"/>
 <ObjectIntersectionOf>
  <Class IRI="Person"/>
  <ObjectComplementOf>
   <Class IRI="Parent"/>
  </ObjectComplementOf>
```

```
 </ObjectIntersectionOf>
</EquivalentClasses>

<EquivalentClasses>
 <Class IRI="Mother"/>
 <ObjectIntersectionOf>
 <ObjectSomeValuesFrom>
  <ObjectProperty IRI="hasChild"/>
  <Class IRI="Person"/>
 </ObjectSomeValuesFrom>
 <Class IRI="Woman"/>
 </ObjectIntersectionOf>
</EquivalentClasses>

<EquivalentClasses>
 <Class IRI="Father"/>
 <ObjectIntersectionOf>
 <ObjectSomeValuesFrom>
  <ObjectProperty IRI="hasChild"/>
  <Class IRI="Person"/>
 </ObjectSomeValuesFrom>
 <Class IRI="Man"/>
 </ObjectIntersectionOf>
</EquivalentClasses>

<EquivalentClasses>
 <Class IRI="Parent"/>
 <ObjectUnionOf>
  <Class IRI="Mother"/>
  <Class IRI="Father"/>
 </ObjectUnionOf>
</EquivalentClasses>

<ObjectPropertyDomain>
 <ObjectProperty IRI="hasWife"/>
 <Class IRI="Man"/>
</ObjectPropertyDomain>

<ObjectPropertyRange>
 <ObjectProperty IRI="hasWife"/>
 <Class IRI="Woman"/>
</ObjectPropertyRange>

<EquivalentClasses>
 <Class IRI="NarcisticPerson"/>
 <ObjectHasSelf>
  <ObjectProperty IRI="loves"/>
 </ObjectHasSelf>
</EquivalentClasses>

<EquivalentObjectProperties>
<ObjectProperty IRI="hasDaughter"/>
<ObjectProperty IRI="hasFemaleChild"/>
</EquivalentObjectProperties>

<SubObjectPropertyOf>
 <ObjectProperty IRI="hasDaughter"/>
 <ObjectProperty IRI="hasChild"/>
</SubObjectPropertyOf>
```

```
<SubObjectPropertyOf>
 <ObjectProperty IRI="hasSon"/>
 <ObjectProperty IRI="hasChild"/>
</SubObjectPropertyOf>

<DisjointObjectProperties>
 <ObjectProperty IRI="hasSon"/>
 <ObjectProperty IRI="hasDaughter"/>
</DisjointObjectProperties>

<InverseObjectProperties>
 <ObjectProperty IRI="hasParent"/>
 <ObjectProperty IRI="hasChild"/>
</InverseObjectProperties>

<FunctionalObjectProperty>
 <ObjectProperty IRI="hasHusband"/>
</FunctionalObjectProperty>

<InverseFunctionalObjectProperty>
 <ObjectProperty IRI="hasHusband"/>
</InverseFunctionalObjectProperty>

<ReflexiveObjectProperty>
 <ObjectProperty IRI="hasRelative"/>
</ReflexiveObjectProperty>

<IrreflexiveObjectProperty>
 <ObjectProperty IRI="parentOf"/>
</IrreflexiveObjectProperty>

<SymmetricObjectProperty>
 <ObjectProperty IRI="hasRelative"/>
</SymmetricObjectProperty>

<AsymmetricObjectProperty>
 <ObjectProperty IRI="hasChild"/>
</AsymmetricObjectProperty>

<TransitiveObjectProperty>
 <ObjectProperty IRI="hasAncestor"/>
</TransitiveObjectProperty>

<SubDataPropertyOf>
 <DataProperty IRI="hasFamilyName"/>
 <DataProperty IRI="hasName"/>
</SubDataPropertyOf>

<DisjointDataProperties>
 <DataProperty IRI="hasAge"/>
 <DataProperty IRI="hasID"/>
</DisjointDataProperties>

<EquivalentDataProperties>
 <DataProperty IRI="hasAge"/>
 <DataProperty IRI="age"/>
</EquivalentDataProperties>
```

```
<DataPropertyDomain>
 <DataProperty IRI="hasAge"/>
 <Class IRI="Person"/>
</DataPropertyDomain>

<DataPropertyRange>
 <DataProperty IRI="hasAge"/>
 <Datatype IRI="xsd:nonNegativeInteger"/>
</DataPropertyRange>

<FunctionalDataProperty>
 <DataProperty IRI="hasAge"/>
</FunctionalDataProperty>

<HasKey>
 <Class IRI="Person"/>
 <dataProperty IRI="hasID"/>
</HasKey>

<DataPropertyAssertion>
 <DataProperty IRI="hasID"/>
 <NamedIndividual IRI="John"/>
 <Literal datatypeIRI="xsd:nonNegativeInteger">123400001</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasID"/>
 <NamedIndividual IRI="Mary"/>
 <Literal datatypeIRI="xsd:nonNegativeInteger">123400002</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasID"/>
 <NamedIndividual IRI="Liz"/>
 <Literal datatypeIRI="xsd:nonNegativeInteger">123400003</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasID"/>
 <NamedIndividual IRI="Amy"/>
 <Literal datatypeIRI="xsd:nonNegativeInteger">123400004</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasID"/>
 <NamedIndividual IRI="Tom"/>
 <Literal datatypeIRI="xsd:nonNegativeInteger">123400005</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasID"/>
 <NamedIndividual IRI="Max"/>
 <Literal datatypeIRI="xsd:nonNegativeInteger">123400006</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasID"/>
 <NamedIndividual IRI="Alice"/>
 <Literal datatypeIRI="xsd:nonNegativeInteger">123400007</Literal>
```

```
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasFamilyName"/>
 <NamedIndividual IRI="John"/>
 <Literal datatypeIRI="xsd:string">Parker</Literal>
</DataPropertyAssertion>

<SubClassOf>
 <DataHasValue>
  <DataProperty IRI="hasFamilyName"/>
  <Literal datatypeIRI="xsd:string">"Parker"</Literal>
 </DataHasValue>
 <Class IRI="Person"/>
</SubClassOf>

<DataPropertyAssertion>
 <Annotation>
  <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
  <Literal datatypeIRI="xsd:string">ID001</Literal>
 </Annotation>
 <DataProperty IRI="hasFamilyName"/>
 <NamedIndividual IRI="Max"/>
 <Literal datatypeIRI="xsd:string">Evans</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasAge"/>
 <NamedIndividual IRI="John"/>
 <Literal datatypeIRI="xsd:integer">51</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasAge"/>
 <NamedIndividual IRI="Mary"/>
 <Literal datatypeIRI="xsd:integer">50</Literal>
</DataPropertyAssertion>

<NegativeDataPropertyAssertion>
 <DataProperty IRI="hasAge"/>
 <NamedIndividual IRI="Mary"/>
 <Literal datatypeIRI="xsd:integer">53</Literal>
</NegativeDataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasAge"/>
 <NamedIndividual IRI="Liz"/>
 <Literal datatypeIRI="xsd:integer">25</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasAge"/>
 <NamedIndividual IRI="Amy"/>
 <Literal datatypeIRI="xsd:integer">21</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasAge"/>
 <NamedIndividual IRI="Max"/>
```

```
 <Literal datatypeIRI="xsd:integer">26</Literal>
</DataPropertyAssertion>

<DataPropertyAssertion>
 <DataProperty IRI="hasAge"/>
 <NamedIndividual IRI="Alice"/>
 <Literal datatypeIRI="xsd:integer">2</Literal>
</DataPropertyAssertion>

<SameIndividual>
 <NamedIndividual IRI="Liz"/>
 <NamedIndividual IRI="Liza"/>
</SameIndividual>

<DifferentIndividuals>
 <NamedIndividual IRI="John"/>
 <NamedIndividual IRI="Tom"/>
</DifferentIndividuals>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasWife"/>
 <NamedIndividual IRI="John"/>
 <NamedIndividual IRI="Mary"/>
</ObjectPropertyAssertion>

<ClassAssertion>
 <ObjectMinCardinality cardinality="1">
  <ObjectProperty IRI="hasChild"/>
  <Class IRI="Parent"/>
 </ObjectMinCardinality>
 <NamedIndividual IRI="John"/>
</ClassAssertion>

<ClassAssertion>
 <ObjectExactCardinality cardinality="2">
  <ObjectProperty IRI="hasChild"/>
  <Class IRI="Parent"/>
 </ObjectExactCardinality>
 <NamedIndividual IRI="John"/>
</ClassAssertion>

<ClassAssertion>
 <ObjectMaxCardinality cardinality="3">
  <ObjectProperty IRI="hasChild"/>
  <Class IRI="Parent"/>
 </ObjectMaxCardinality>
 <NamedIndividual IRI="John"/>
</ClassAssertion>

<EquivalentClasses>
 <Class IRI="JohnsChildren"/>
 <ObjectHasValue>
  <ObjectProperty IRI="hasParent"/>
  <NamedIndividual IRI="John"/>
 </ObjectHasValue>
</EquivalentClasses>

<EquivalentClasses>
 <Class IRI="JohnsChildren"/>
```

```
 <ObjectOneOf>
  <NamedIndividual IRI="Liz"/>
  <NamedIndividual IRI="Tom"/>
  <NamedIndividual IRI="Amy"/>
 </ObjectOneOf>
</EquivalentClasses>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasRelative"/>
 <NamedIndividual IRI="John"/>
 <NamedIndividual IRI="Alice"/>
</ObjectPropertyAssertion>

<ClassAssertion>
 <Class IRI="Mother"/>
 <NamedIndividual IRI="Mary"/>
</ClassAssertion>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasChild"/>
 <NamedIndividual IRI="Mary"/>
 <NamedIndividual IRI="Liz"/>
</ObjectPropertyAssertion>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasSon"/>
 <NamedIndividual IRI="Mary"/>
 <NamedIndividual IRI="Tom"/>
</ObjectPropertyAssertion>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasDaughter"/>
 <NamedIndividual IRI="Mary"/>
 <NamedIndividual IRI="Amy"/>
</ObjectPropertyAssertion>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasAncestor"/>
 <NamedIndividual IRI="Mary"/>
 <NamedIndividual IRI="Alice"/>
</ObjectPropertyAssertion>

<ClassAssertion>
 <DataMaxCardinality cardinality="2">
  <DataProperty IRI="hasName"/>
 </DataMaxCardinality>
 <NamedIndividual IRI="Liz"/>
</ClassAssertion>

<ClassAssertion>
 <DataMinCardinality cardinality="1">
  <DataProperty IRI="hasName"/>
 </DataMinCardinality>
 <NamedIndividual IRI="Liz"/>
</ClassAssertion>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasBrother"/>
 <NamedIndividual IRI="Liz"/>
```

```
 <NamedIndividual IRI="Tom"/>
</ObjectPropertyAssertion>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasHusband"/>
 <NamedIndividual IRI="Liz"/>
 <NamedIndividual IRI="Max"/>
</ObjectPropertyAssertion>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="ParentOf"/>
 <NamedIndividual IRI="Liz"/>
 <NamedIndividual IRI="Alice"/>
</ObjectPropertyAssertion>

<ClassAssertion>
 <Class IRI="Teenager"/>
 <NamedIndividual IRI="Tom"/>
</ClassAssertion>

<ClassAssertion>
 <ObjectAllValuesFrom>
  <ObjectProperty IRI="hasSibling"/>
  <Class IRI="Woman"/>
 </ObjectAllValuesFrom>
 <NamedIndividual IRI="Tom"/>
</ClassAssertion>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasSibling"/>
 <NamedIndividual IRI="Tom"/>
 <NamedIndividual IRI="Liz"/>
</ObjectPropertyAssertion>

<ObjectPropertyAssertion>
 <ObjectProperty IRI="hasSister"/>
 <NamedIndividual IRI="Tom"/>
 <NamedIndividual IRI="Amy"/>
</ObjectPropertyAssertion>

<ClassAssertion>
 <Class IRI="ChildlessPerson"/>
 <NamedIndividual IRI="Amy"/>
</ClassAssertion>

<ClassAssertion>
 <DataExactCardinality cardinality="1">
  <DataProperty IRI="hasName"/>
 </DataExactCardinality>
 <NamedIndividual IRI="Max"/>
</ClassAssertion>

<NegativeObjectPropertyAssertion>
 <ObjectProperty IRI="hasWife"/>
 <NamedIndividual IRI="Max"/>
 <NamedIndividual IRI="Mary"/>
</NegativeObjectPropertyAssertion>

<ClassAssertion>
```

```
    <Class IRI="Father"/>
    <NamedIndividual IRI="Max"/>
   </ClassAssertion>

   <ObjectPropertyAssertion>
    <Annotation>
     <AnnotationProperty IRI="http://omdoc.org/identifier#id"/>
     <Literal datatypeIRI="xsd:string">ID002</Literal>
    </Annotation>
    <ObjectProperty IRI="hasDaughter"/>
    <NamedIndividual IRI="Max"/>
    <NamedIndividual IRI="Alice"/>
   </ObjectPropertyAssertion>

</Ontology>
```