Saarland University

Faculty of Natural Sciences and Technology I

Department of Computer Science

Master thesis

Semantic Gap Detection in Learning Object Metadata

submitted by

Isaac Alpízar Chacón

submitted

26.07.2012

Supervisor

Prof. Dr. Jörg Siekmann

Advisor

Dr. Sergey Sosnovsky

Reviewers

Prof. Dr. Jörg Siekmann

Priv.-Doz. Dr. Christoph Igel

Eidesstattliche Erklrung

Ich erklre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbststndig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverstndniserklrung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit verffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Isaac Alpízar Chacón

Saarbrücken, 26.07.2012

a mi

MADRE

por su apoyo y amor incondicional

——o——

to my

MOTHER

for all her unconditional support and love

Acknowledgements

I would like to express my gratitude to my advisor, Dr. Sergey Sosnovsky, for his advice, constant support, patience, and kindness. He has guided me since I came to Saarland University, and trusted me with this project from the beginning. Even when I had committed mistakes, he gave me constructive criticism instead of just telling me how bad I was doing a certain task. I am grateful to Prof. Dr. Jörg Siekmann, for being my supervisor and having provided me with the opportunity and guidance to pursue this thesis. I also want to thank him for first introduced me Saarland University, and for gave me his support to come to Germany. I would further like to thank all the members of the ActiveMath group, which welcomed and helped me since my first day in Saarbrüken.

I also wish to thank Mario Chacón Rivas, from the Costa Rica Institute of Technology, because he trusted and supported me since our first project together, and continued doing it with my master. From the Costa Rica Institute of Technology, I must also thank everyone whom one way or the other helped me to be here, and give me the possibility to accomplish this important milestone in my life.

Additionally, I want to thank the DAAD for partially support my studies, and to support the studies of many other people from Costa Rica.

Finally, without the support of my family and friends, this journey would not have been possible. They give me their support every day, and when tiredness appears, they always have encouraging words.

Abstract

Metadata is widely used to describe properties of different digital collections and their elements. In e-learning, metadata is used to define properties of individual learning objects and collections of educational content. However, metadata annotation is a difficult task; therefore, content collections contain gaps and inconsistencies. Resolution of such gaps is a challenging and important problem, because erroneous instructional material presented to a student can lead to confusion, frustration, lack of motivation and, essentially, poor learning.

This thesis aims to provide a mechanism to validate the correctness of the learning object metadata. The key feature of this mechanism is the use of formal ontologies to account to underline semantic constraints of learning object metadata. The proposed approach consists of four steps: *1*) conversion of the learning object metadata to Web Ontology Language format; *2*) detection of inconsistencies in the content using a semantic reasoner; *3*) isolation of the specific gaps in the metadata; and *4*) generation of human-readable explanations for solving the gaps.

The proposed approach has been implemented as a tool to detect inconsistencies in the learning content of ActiveMath. This tool can detect different types of gaps in the content, and results from the evaluation have shown that it can effectively detect gaps in learning collections.

Table of Contents

Page

Acknowledgements . . . vii

Abstract . . . ix

Table of Contents . . . xi

List of Tables . . . xiv

List of Figures . . . xv

Chapter

1 Introduction . . . 1

1.1 Context . . . 1

1.2 Motivation . . . 2

1.3 Approach . . . 3

1.4 Definitions . . . 4

1.4.1 Ontology . . . 4

1.4.2 The OWL 2 Web Ontology Language (OWL 2) . . . 4

1.4.3 Semantic Reasoner . . . 5

1.4.4 Extensible Stylesheet Language Transformations (XSLT) . . . 5

1.4.5 Metadata . . . 5

1.4.6 Metadata Gap . . . 5

1.4.7 Intelligent Tutoring Systems (ITS) . . . 6

1.5 Thesis Structure . . . 6

2 ActiveMath . . . 7

2.1 General Architecture . . . 7

2.2 Knowledge Representation . . . 8

2.3 Metadata . . . 9

2.4 Content Presentation . . . 11

2.5 Conclusion 12
3 Background and Related Work 14
3.1 Semantic Web Technologies Supporting Authoring For ITS 14
3.1.1 Guidelines for formalizing instruction 16
3.1.2 Metadata validation 18
3.1.3 Metadata generation 20
3.1.4 Learning object discovery 24
3.1.5 Summary 24
3.2 Background: Gap Detection in ActiveMath 27
3.2.1 Types of gaps 27
3.2.2 Gap detection 27
3.2.3 Limitations 29
4 Approach 31
4.1 Metadata Requirements Definition 31
4.2 Metadata Validation 32
4.2.1 Conversion of metadata to OWL2 33
4.2.2 Detection of ontology inconsistencies 35
4.2.3 Isolation of causing axioms 36
4.2.4 Generation of verbal explanations 38
5 Implementation 41
5.1 General Description 41
5.2 Transformation 42
5.2.1 XSLT stylesheet 42
5.3 Validation 47
5.3.1 Pre-processing 48
5.3.2 Ontology inconsistencies 49
5.3.3 Isolation of axioms 49
5.3.4 Verbal explanation 50

6 Evaluation and Results 56
6.1 Gaps in ActiveMath 56
6.2 Effectiveness 58
6.2.1 Procedure 58
6.2.2 Results 60
6.3 Discussion 62
7 Conclusion and Future Work 64
7.1 Conclusion 64
7.2 Future Work 65
References 66

List of Tables

5.1 Verbal explanations for gaps 52

6.1 Number of gaps by collection 57

6.2 Detailed number of gaps by type 57

6.3 Effect in the number of gaps by type of changes 60

List of Figures

1.1 Problems of metadata annotation 3

2.1 Architecture of ActiveMath 8
2.2 Knowledge items of ActiveMath 10
2.3 OMDoc code for an example in ActiveMath 12
2.4 Basic course in ActiveMath 13

3.1 Related work classification 1 15
3.2 Interface of an Ontology based Authoring Tool 17
3.3 ASPIRE's Ontology workspace 19
3.4 Generation of suggested metadata values through semantical similarity . . 21
3.5 Suggestions and restrictions of metadata in LessonMapper2 23
3.6 Related work classification 2 26
3.7 Structural gap detection 28
3.8 Linguistic gap detection 29
3.9 Didactic gap detection 29

4.1 ActiveMath metadata schema 33
4.2 Semantic gap detection procedure 33
4.3 Example of XSLT transformation 35
4.4 Inconsistency example 37
4.5 Isolation of axioms causing an ontology to be inconsistent 38
4.6 Example of a verbal explanation 39

5.1 XSLT rule for the *omdoc* element 43
5.2 XSLT rule for *examples* 45

5.3 Part of the XSLT rule matching the *relation* element in OMDoc 47
5.4 Part of the XSLT rule matching the `for` attribute in OMDoc 48

6.1 Comparison process between versions of a learning collection 58
6.2 Chart of the effect in the number of gaps by type of changes 61
6.3 Changes in the number of gaps in the Version 1 of the collection 61
6.4 Changes in the number of gaps in the Version 2 of the collection 62

Chapter 1

Introduction

1.1 Context

Metadata (see def. 1.4.5, page 5) for digital resources emerged as a way of describing properties of different digital collections and their elements. Metadata is necessary due to the ever-increasing number of digital collections and resources that are created by e-learning systems, digital libraries, knowledge repositories and other systems and organizations today. The usage of high quality metadata in digital resources supports long-term preservation and usage, resource discovery, use-based retrieval, within-collection organization, retrieval of non-textual media, and interoperability among collections [17].

In education, specialized metadata standards have been created to define pedagogical properties of learning objects, and promote both interoperability and sharing of those resources. Many collections of learning content have been annotated with metadata, such as Connexions[1], Intergeo[2], Ariadne[3], and Curriki[4]. Many e-learning systems use metadata to describe the learning content and adapt it to the students. For example, the ActiveMath learning environment uses several collections of mathematical teaching material that consist together of more than 12 000 learning objects, which all are annotated with metadata. Depending on how detailed is the used metadata schema, the annotation of learning resources with metadata makes the authoring process a costly task in terms of human effort and time.

[1] `http://cnx.org/`
[2] `http://i2geo.net/`
[3] `http://www.ariadne-eu.org/`
[4] `http://www.curriki.org`

1.2 Motivation

Quality of learning objects metadata can be affected due to several reasons:

1. Metadata needs human judgment to be created; therefore, it is constructed manually. People can make mistakes which reflect in inconsistencies in metadata. Those inconsistencies can arise due to differences in the judgment of the same author over time, and because different authors may make varied judgments in metadata annotation [5].

2. Metadata annotation is a time-consuming task. For example, a typical course may be composed of several dozen content objects, each of them annotated with various metadata, which would result in a combined set of 1 000 to 5 000 metadata values [16].

3. Metadata is often created by authors who lack technical expertise, making the metadata annotation a difficult task. Authors need to have the necessary domain, pedagogical, knowledge engineering and metadata standards expertise to annotate the resources with the correct metadata. For instance, just the IEEE Learning Object Metadata standard specifies about 70 different metadata elements, from which many of them are complex to instantiate, such as *coverage*, *typical learning time*, and *difficulty* [21].

4. Existing authoring tools provide limited support of metadata authoring. They are designed by experts for experts and focus on functionality rather than usability.

5. Many learning collections have been created in a collaborative way, where several authors modify each other's work, producing a potential source of gaps. For example, in ActiveMath several dozen of authors have contributed in the creation of the learning collections.

Figure 1.1 shows a summary of the problems of metadata annotation.

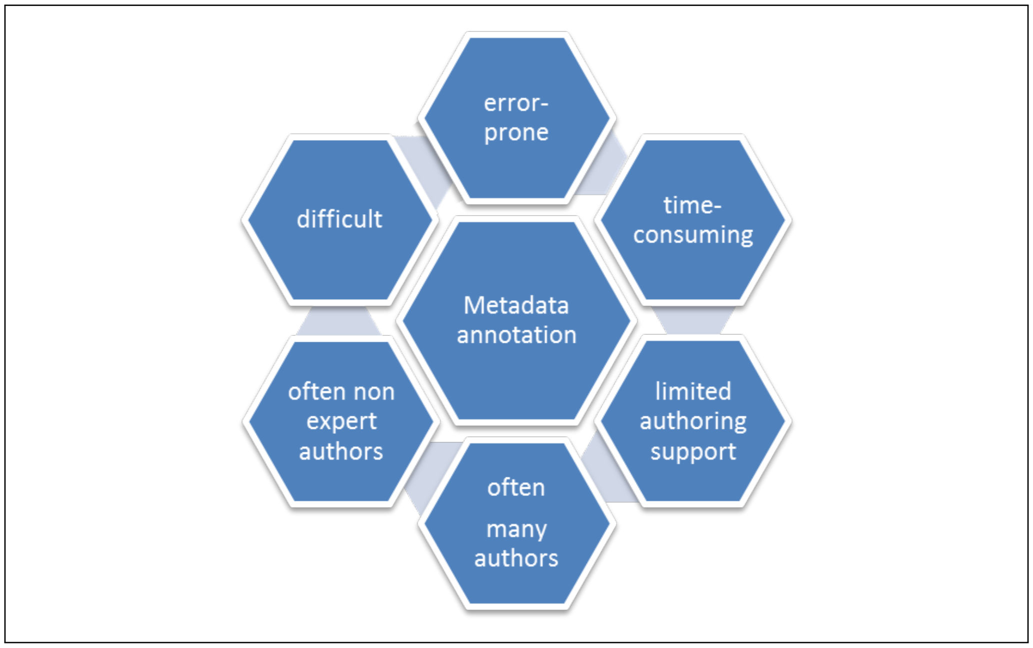


Figure 1.1: Problems of metadata annotation

Poor quality of metadata not only limits the capacity to share and reuse the learning objects, but also affects the quality of adaptive learning systems presenting these learning objects to students. Metadata inconsistencies and gaps (see def. 1.4.6, page 5) may be the cause of various problems from system crashing to ineffective learning experience. In the case of ActiveMath, which uses a course generation component that presents personalized content to the students, if the metadata for the available learning objects is poor in quality, erroneous instructional material can be presented to the student. This mismatch between the formal requirements of metadata usage in the target application (e.g. adaptive systems, digital libraries) and the loose control of metadata authoring justify the creation of a tool that can verify the learning object metadata after or during its creation.

1.3 Approach

This thesis presents an intelligent and rigorous mechanism for detecting metadata gaps in the collections of learning content. The approach uses Semantic Web representation technologies to formalize metadata annotation and verify the correctness of the metadata.

The metadata annotation is formalized by creating an OWL2 ontology (see def. 1.4.1, and 1.4.2, page 4) that describes all the requirements and constraints that need to be satisfied in the metadata. After the learning content has been created and annotated with metadata, a verification mechanism detects the existing inconsistencies according to the metadata OWL2 schema. The mechanism first converts the learning object metadata into an OWL2 representation, from an XML-based language using XSLT transformations (see def. 1.4.4, page 5). Then, it detects logical conflicts using a standard Semantic Web reasoner (see def. 1.4.3, page 5) and the formal metadata OWL2 schema. Finally, human-readable explanations for resolving the gaps are generated and presented to the authors.

1.4 Definitions

1.4.1 Ontology

In terms of the Semantic Web, an ontology is "a document that formally defines terms and relations among them in a sharable format" [3]. In the case of Adaptive Educational Systems, "ontologies primarily serve as domain models; they specify the structure of the subject domain, formally represent its main entities and the relations between them" [28].

1.4.2 The OWL 2 Web Ontology Language (OWL 2)

The OWL 2 (Web Ontology Language) is "an ontology language for the Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents. OWL 2 ontologies can be used along with information written in RDF, and OWL 2 ontologies themselves are primarily exchanged as RDF documents" [6].

1.4.3 Semantic Reasoner

A semantic reasoner[5] is a program that can perform reasoning tasks, from a set of asserted facts or axioms. In the Semantic Web, the facts are usually represented using RDFS, OWL or any other ontology language. Reasoners are important because knowledge in an ontology might not be explicit and a reasoner is required to deduce implicit knowledge in order to get the correct query results.

1.4.4 Extensible Stylesheet Language Transformations (XSLT)

XSLT[6] is a declarative, XML-based language for transforming XML documents into XHTML documents or to other XML documents. Transformation rules are defined in an XSLT stylesheet, which is then applied to the original document to produce a new document.

1.4.5 Metadata

Metadata is "structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource. Metadata is often called data about data or information about information" [23]. In the context of learning, it describes learning objects and similar digital resources used to support learning. Adaptive systems use metadata to reason about digital resources in terms of their properties and personalize users' access to them accordingly.

1.4.6 Metadata Gap

A metadata gap occurs when a metadata element is not used by an author according to the formal requirements defined in the metadata schema. It can be a wrong value for the element, a missing element, or a wrong element for a learning object of a particular type.

[5] `http://www.w3.org/2001/sw/wiki/Category:Reasoner,http://owlapi.sourceforge.net/reasoners.html`

[6] `http://www.w3.org/TR/xslt/`

1.4.7 Intelligent Tutoring Systems (ITS)

An Intelligent Tutoring System is a “computer system that organizes learning process in a student-adaptive way by presenting to students individualized sequences of learning material, providing them with personalized help while they are solving learning problems, and/or using other forms of intelligent learning support tailored for each individual student” [27].

1.5 Thesis Structure

This thesis is organized as follows. Chapter 2 introduces ActiveMath, a web-based intelligent tutoring system for mathematics, in which the proposed approach has been implemented. Chapter 3 continues with a review of related research that uses Semantic Web technologies to support learning content authoring. This chapter also describes a previous tool created in ActiveMath for gap detection. Chapter 4 presents all the details and steps of the proposed approach, along with the necessary technologies. The implementation details of the proposed approach as a tool to detect gaps in the content of ActiveMath are presented in Chapter 5. Chapter 6 describes an evaluation carried out using several learning collections of ActiveMath to test the correct behavior of the tool and discusses the obtained results. Finally, Chapter 7 presents the conclusion and discusses future work.

Chapter 2

ActiveMath

The approach proposed in this thesis has been implemented within the framework of ActiveMath, a web-based intelligent tutoring system for mathematics, which uses Semantic Web features to enhance the student's learning [18]. This chapter introduces ActiveMath, explains its architecture and outlines the most important features.

2.1 General Architecture

ActiveMath has a client-server architecture, as shown in Figure 2.1. It has typical components of an ITS, such as domain model, a student model, an exercise system, and pedagogical modules comprising course generator, tutorial strategies, and feedback generators.

The domain model is encoded implicitly in the content stored in the *content base(s)*. This design allows the system to be open to new content, in such a way that content can be added/modified by a community of authors. Because of that, ActiveMath can take care of content model changes without the necessity to re-engineer the whole knowledge base.

Information about the knowledge and the performance of the students is recorded in the student model (*competencies* and *history*). The student model provides information to other components such as the *course generator* and *exercise system*. Competencies of the student in ActiveMath can be modeled using different competency frameworks. For instance, the PISA[1] specification for mathematics 'competencies' includes `think`, `argue`, `model`, `solve`, `represent`, `language`, `tools`.

The *exercise system* generates interactive exercises and provides feedback and hints

[1]OECD Programme for International Student Assessment (PISA). `http://www.pisa.oecd.org/`

to the students. Computer algebra systems (CAS's) are used to generate flag feedback (correct/incorrect feedback), as well correct solutions to given problems. More powerful diagnosis, such as errors in a solution, the student's solution strategy, and irrelevant steps, can be obtained from a domain reasoner.

The *course generator* uses information about learning objects, the learners and their learning goals to generate an adapted sequence of learning objects that supports the students in achieving their goals. Hence it communicates with the student *model* and the *content base(s)*.

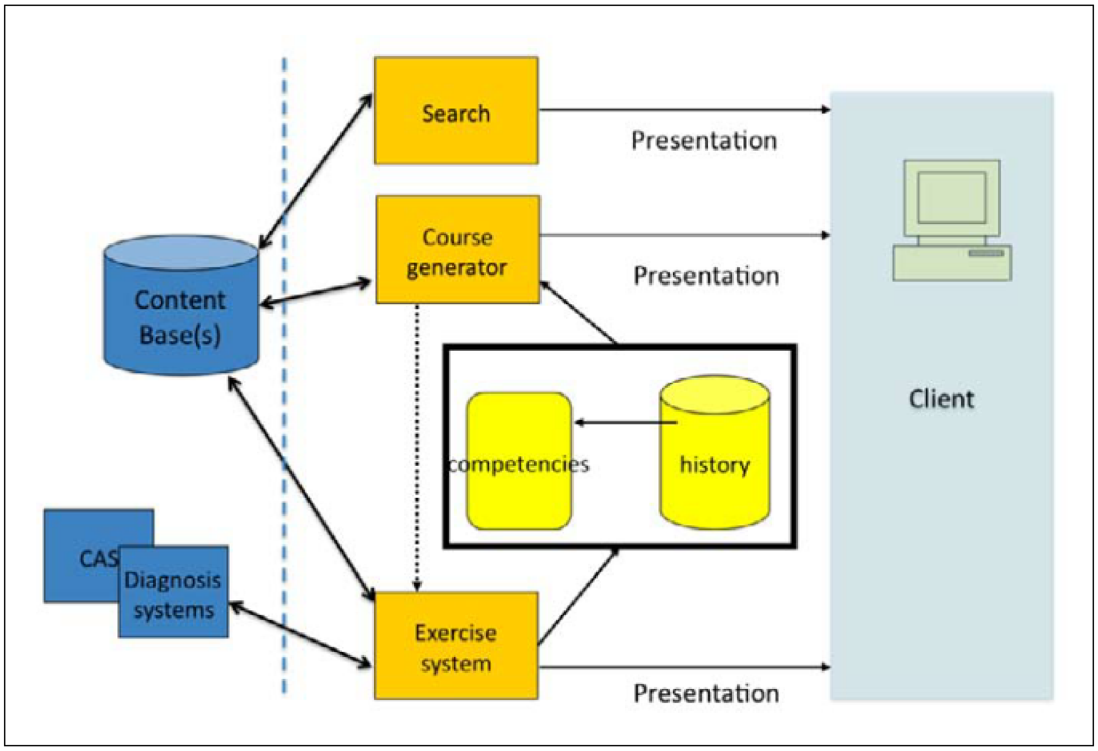


Figure 2.1: Architecture of ActiveMath (picture taken from [18])

2.2 Knowledge Representation

The knowledge representation in ActiveMath is based on the Web standard OMDoc (Open Mathematical Documents). It is an open markup language, an XML dialect, and a data model for mathematical documents on the web [14, 15]. OMDoc can take existing knowledge and annotate it with the information required to retrieve and combine it automatically. Furthermore, it is extensible, which allows new developments in mathematics to be introduced, an essential feature for the growing and complex field of mathematics. The OMDoc format uses and extends OpenMath[2], for representing mathematical formulae and encoding the meaning (semantics) of the symbols, rather than just their visual representation. Additionally, OMDoc allows to enrich mathematical documents with text descriptions in multiple languages, links, and multimedia elements such as Java applets or Adobe Flash animations and videos.

ActiveMath also uses OMDoc to define fine-grained learning objects that are connected to each other by relations and annotated with metadata. ActiveMath supports two principle types of learning objects (or items): concept and satellite items (see Figure 2.2). The first ones express knowledge, such a symbols representing concepts, definitions of these concepts, assertions, axioms, proofs and misconceptions. Satellite items complement and train the related concepts. They can be examples, exercises or instructional texts. This network of learning objects in ActiveMath allows to describe each learning object sufficiently precise for intelligent components to integrate them automatically into the students learning work flow.

2.3 Metadata

Learning objects in ActiveMath are annotated with metadata, which is divided into three main types: descriptive metadata, pedagogic metadata and semantic metadata. A complete reference can be founded in [27].

Descriptive metadata is used to catalog learning objects and facilitate their discovery. This category includes general informations such as *title* of the object, *date* of creation, *author(s)*, *copyright* and *licensing*, and technical characteristics like *language*. The descriptive

[2]OpenMath, an emerging standard for representing mathematical objects with their semantics. `http://www.openmath.org/standard/`

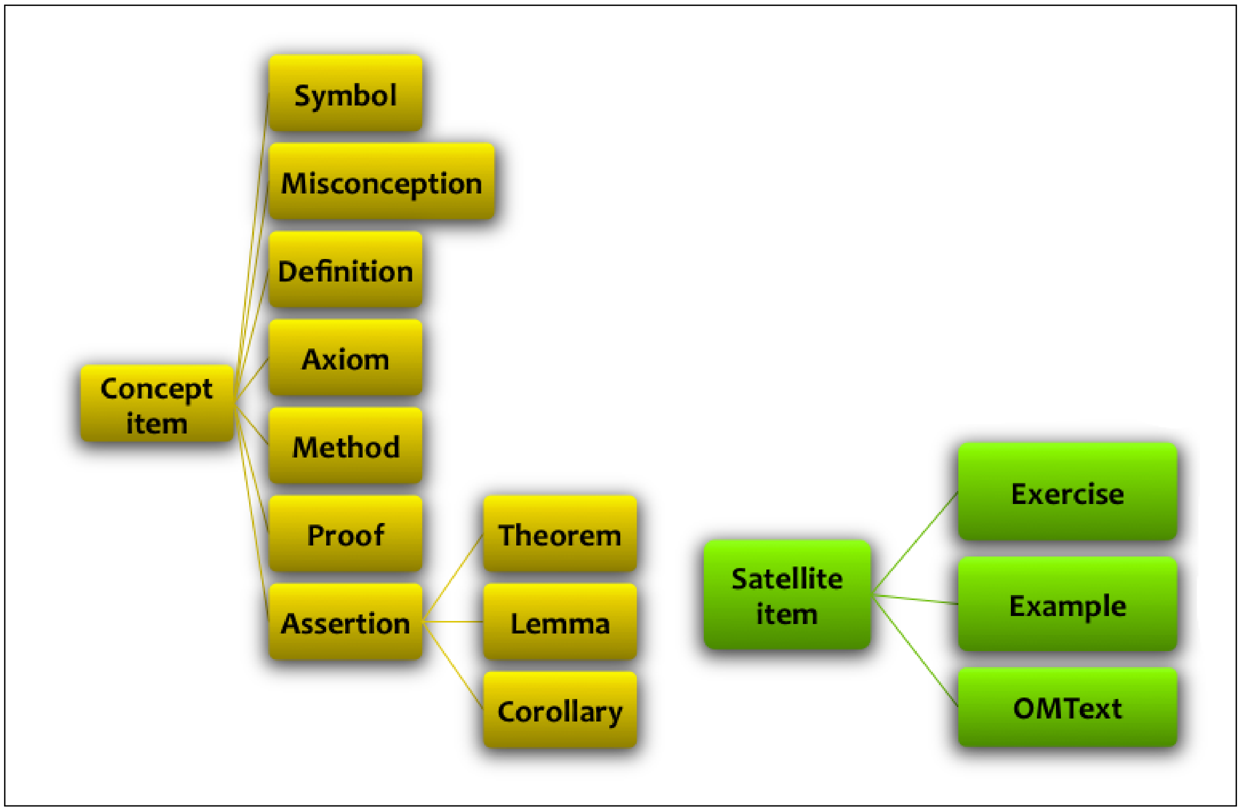


Figure 2.2: Knowledge items of ActiveMath (picture taken from [27])

metadata reuses the standard Dublin Core metadata terms[3].

Pedagogical metadata describes pedagogic properties of the knowledge items (for example, *learning_context*, *difficulty*, *field*, and *abstractness*). It includes terms imported from existing standards such as LOM[4] , IMS[5] , and Ariadne[6]. The right choice of pedagogic metadata is necessary in order for ActiveMath to present to a student learning objects with proper *difficulty*, *competency*, etc.

Semantic metadata defines the types of learning objects, such as *definition*, *theorem*, *exercise*, and the dependency relations between them. For example, the relation *domain_prerequisite* links a learning object with the needed concept in order to start working with

[3]Metadata vocabularies in support of inter operable solutions for discovering and managing resources. `http://dublincore.org/`

[4]IEEE standard for Learning Object Metadata. `http://ltsc.ieee.org/wg12/`

[5]IMS Learning Design Specification. `http://www.imsglobal.org/learningdesign/`

[6]ARIADNE educational metadata recommendation. `http://www.cen-ltso.net/Main.aspx?put=824`

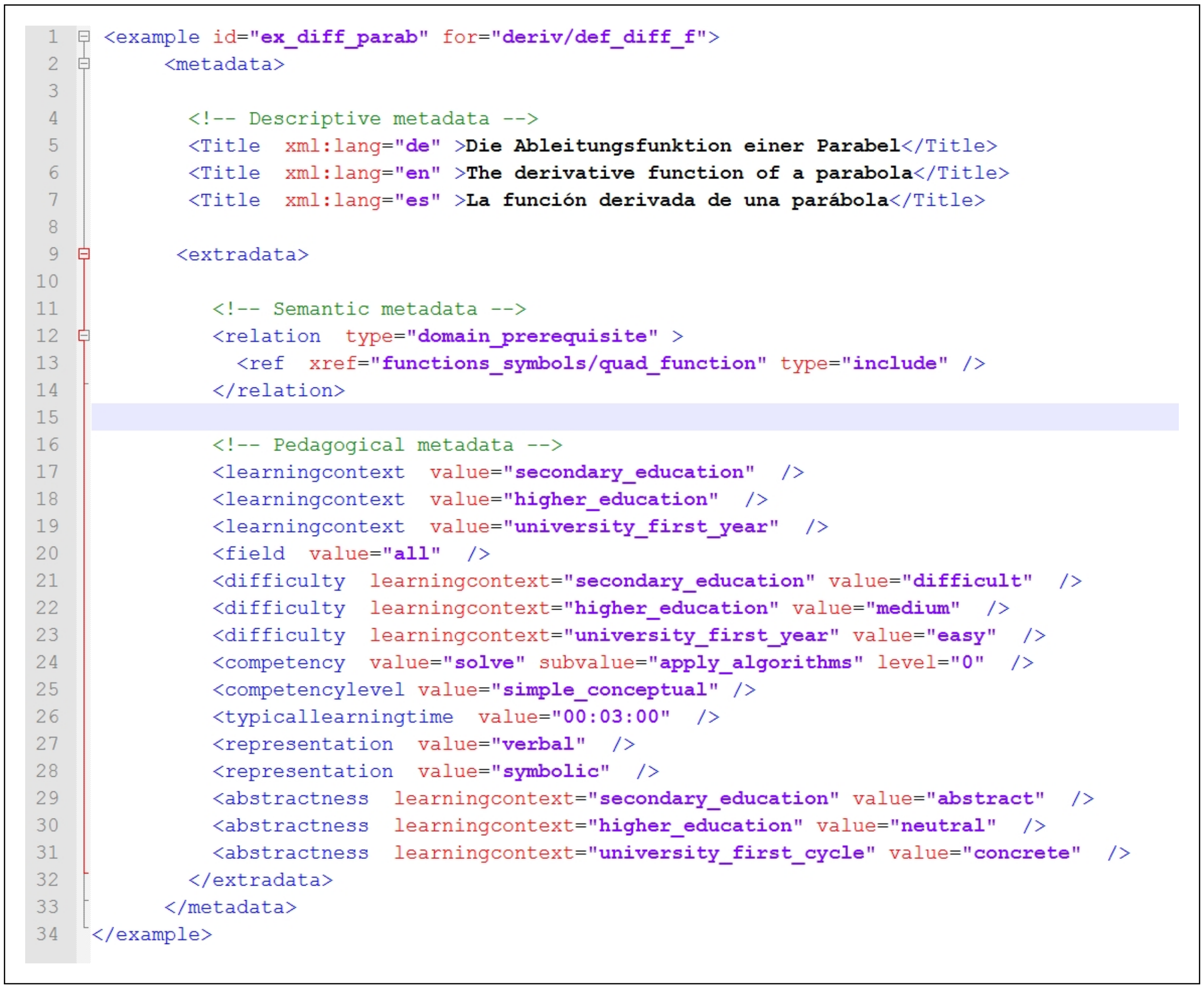
the current knowledge item. The type of the relation defines it semantics, and relations of different types are used to link different learning objects.

Figure 2.3 shows the representation of an example and its metadata using OMDoc. The learning object is part of the LeAM-calculus collection, available in ActiveMath. A learning object starts with the definition of its identifier (`id='ex_diff_parab'`), and the use of the relation *for*, that in this case indicates that the learning object is an example for the object `'deriv/def_diff_f'`, which in its turn is a definition of a symbol. As part of the descriptive metadata, the *title* is defined in several languages. The example is also connected with the *domain_prerequisite* semantic relation to the symbol `'functions_symbols/quad_function'`. Finally, we can see several values for pedagogical metadata, such as *learning_context*, *difficulty*, and *representation.*

2.4 Content Presentation

The learning material in ActiveMath is presented to students in the form of courses. Each course consists of sections, which consist of a series of pages. A page is composed of a set of learning objects; for example, a small page can contain a definition, an example, and two exercises. A student either can take existing courses assembled manually or generate personalized courses. These personalized courses will be generated according to the students state of knowledge, learning goals and will take into account the dependency structure between the learning objects and their pedagogical properties.

Figure 2.4 presents an example of a course in ActiveMath. The student can navigate through the course using the panel on the left. A bar next to every chapter indicates the progress of the student with the learning material of this chapter. In the central panel, the learning content is displayed. The student can click on any learning object to get contextual information about the object (see the right panel).

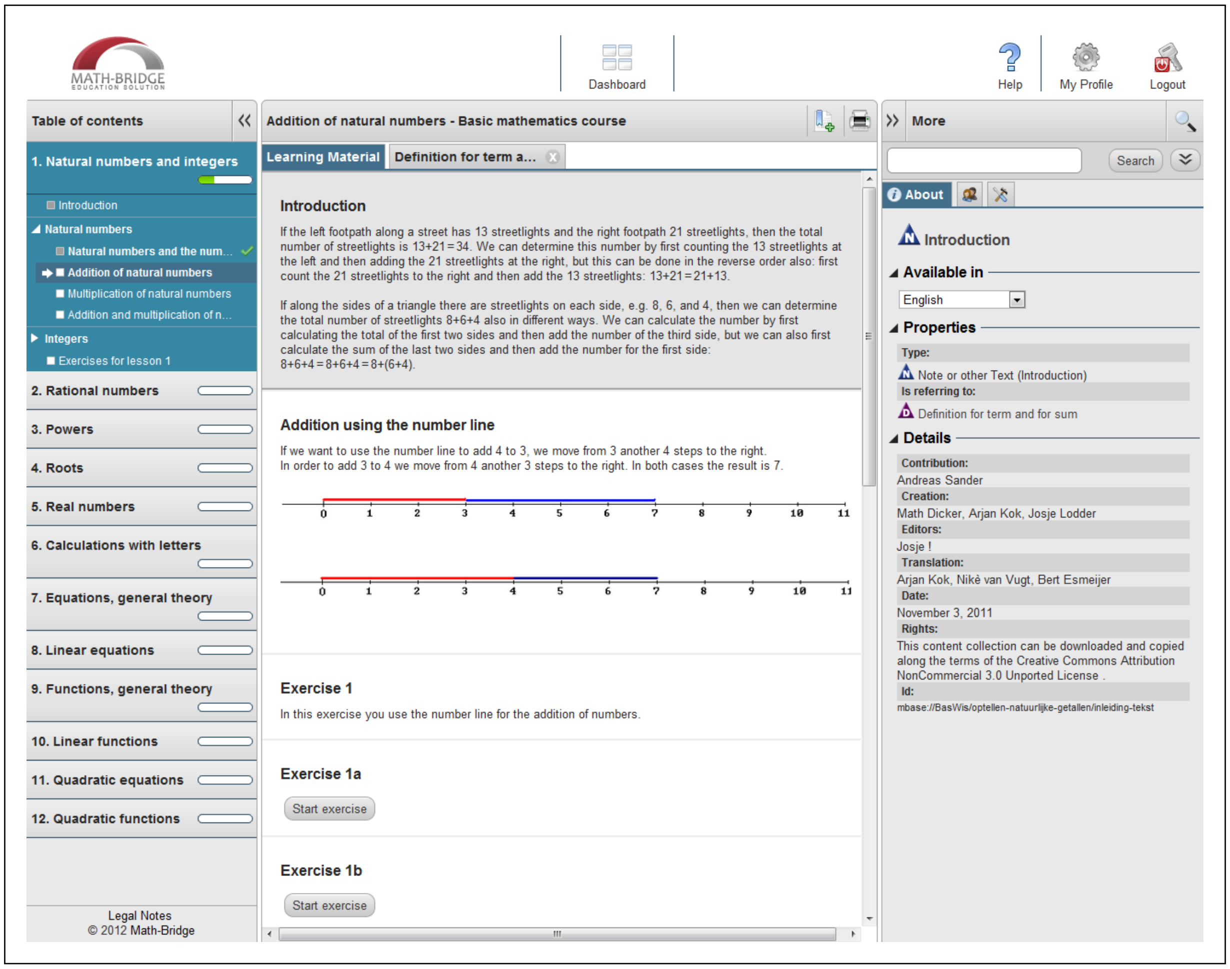


```xml
<example id="ex_diff_parab" for="deriv/def_diff_f">
     <metadata>

       <!-- Descriptive metadata -->
       <Title  xml:lang="de" >Die Ableitungsfunktion einer Parabel</Title>
       <Title  xml:lang="en" >The derivative function of a parabola</Title>
       <Title  xml:lang="es" >La función derivada de una parábola</Title>

      <extradata>

         <!-- Semantic metadata -->
         <relation  type="domain_prerequisite" >
           <ref  xref="functions_symbols/quad_function" type="include" />
         </relation>

         <!-- Pedagogical metadata -->
         <learningcontext  value="secondary_education"  />
         <learningcontext  value="higher_education"  />
         <learningcontext  value="university_first_year"  />
         <field  value="all"  />
         <difficulty  learningcontext="secondary_education" value="difficult"  />
         <difficulty  learningcontext="higher_education" value="medium"  />
         <difficulty  learningcontext="university_first_year" value="easy"  />
         <competency  value="solve" subvalue="apply_algorithms" level="0"  />
         <competencylevel value="simple_conceptual" />
         <typicallearningtime  value="00:03:00"  />
         <representation  value="verbal"  />
         <representation  value="symbolic"  />
         <abstractness  learningcontext="secondary_education" value="abstract"  />
         <abstractness  learningcontext="higher_education" value="neutral"  />
         <abstractness  learningcontext="university_first_cycle" value="concrete"  />
       </extradata>
     </metadata>
</example>
```

Figure 2.3: OMDoc code for an example in ActiveMath

2.5 Conclusion

This chapter described ActiveMath learning environment. As seen before, the system has the typical components of an ITS and additionally, uses advanced features that make it a Semantic Web application. The knowledge representation is based on OMDoc to describe mathematical formulae and define a structure of learning objects. The learning content is annotated with descriptive, pedagogical and semantic metadata. Thanks to such metadata, and the network of learning objects, ActiveMath can present to the students personalized



Addition of natural numbers - Basic mathematics course

Introduction

If the left footpath along a street has 13 streetlights and the right footpath 21 streetlights, then the total number of streetlights is 13+21=34. We can determine this number by first counting the 13 streetlights at the left and then adding the 21 streetlights at the right, but this can be done in the reverse order also: first count the 21 streetlights to the right and then add the 13 streetlights: 13+21=21+13.

If along the sides of a triangle there are streetlights on each side, e.g. 8, 6, and 4, then we can determine the total number of streetlights 8+6+4 also in different ways. We can calculate the number by first calculating the total of the first two sides and then add the number of the third side, but we can also first calculate the sum of the last two sides and then add the number for the first side: 8+6+4=8+6+4=8+(6+4).

Addition using the number line

If we want to use the number line to add 4 to 3, we move from 3 another 4 steps to the right.
In order to add 3 to 4 we move from 4 another 3 steps to the right. In both cases the result is 7.

Exercise 1

In this exercise you use the number line for the addition of numbers.

Exercise 1a

Exercise 1b

Introduction

Available in

English

Properties

Type: Note or other Text (Introduction)

Is referring to: Definition for term and for sum

Details

Contribution: Andreas Sander

Creation: Math Dicker, Arjan Kok, Josje Lodder

Editors: Josje !

Translation: Arjan Kok, Nikè van Vugt, Bert Esmeijer

Date: November 3, 2011

Rights: This content collection can be downloaded and copied along the terms of the Creative Commons Attribution NonCommercial 3.0 Unported License .

Id: mbase://BasWis/optellen-natuurlijke-getallen/inleiding-tekst

Figure 2.4: Basic course in ActiveMath

courses for a better learning experience.

Chapter 3

Background and Related Work

The Gap Detection Tool presented in this thesis uses Semantic Web technologies to facilitate the discovery of gaps in learning object metadata. This chapter provides an overview of related research on the use of Semantic Web technologies for learning content authoring. It also introduces the previous work on Gap Detection Tool developed for ActiveMath, and describes it functionality and limitations.

3.1 Semantic Web Technologies Supporting Authoring For ITS

Over the last decade, several projects have employed Semantic Web technologies to help the authors creating learning content and its metadata. They can be classified into four main categories (see Figure 3.1). *Guidelines for formalizing instruction* introduces tools that support creation of learning scenarios with the use of ontologies encoding pedagogical and domain knowledge. *Metadata validation* focuses on how learning object metadata can be validated using an ontology as metadata schema, or with value restrictions extracted from the semantics of the learning objects. *Metadata generation* presents how using the already existing learning object metadata in repositories, automatic suggestion or creation of such metadata values for new or incomplete learning objects can be supported. Finally, *learning object discovery* introduces how to discover learning objects by classify them using suggestions and value restrictions generated from the content being authored.

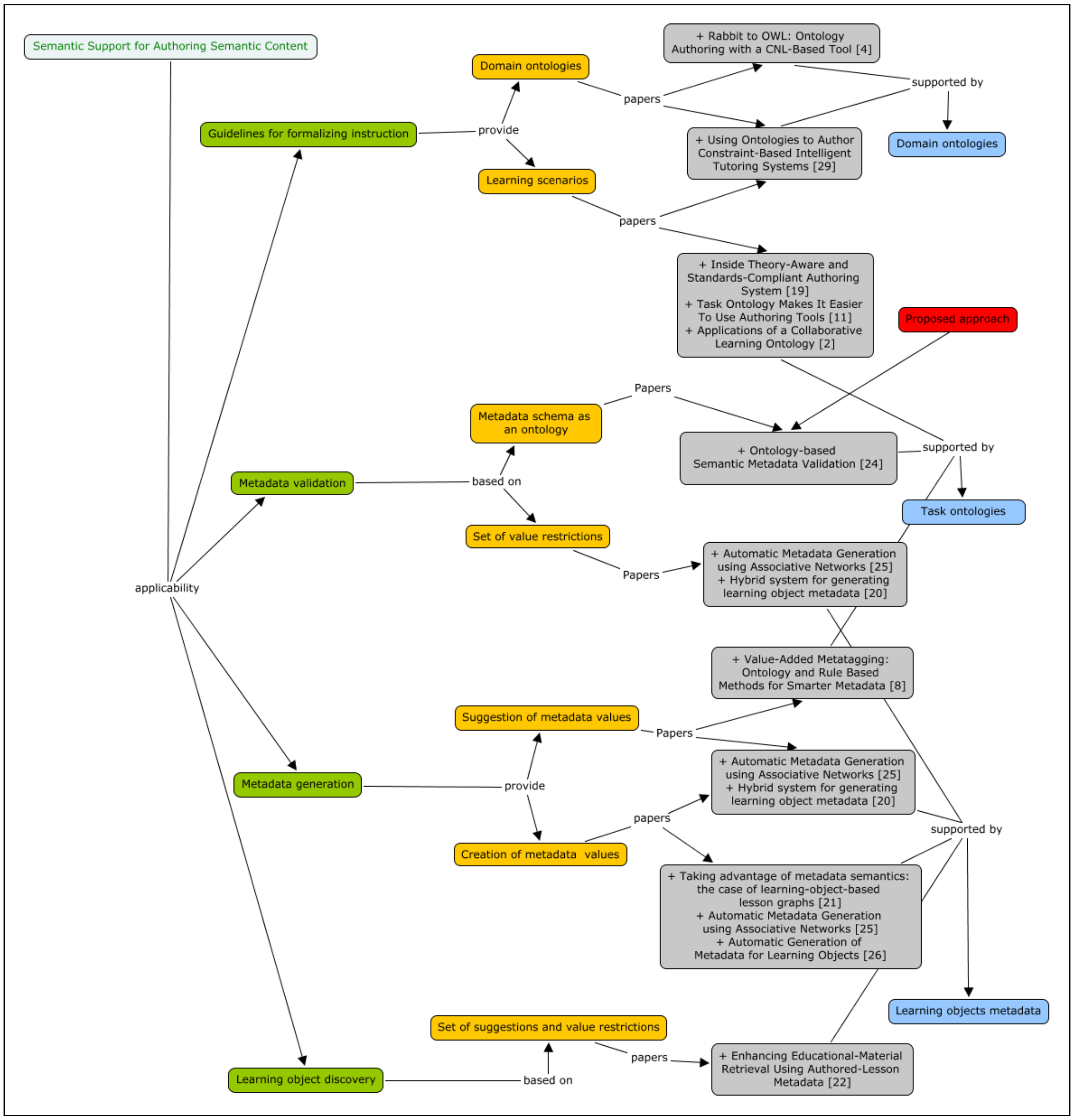
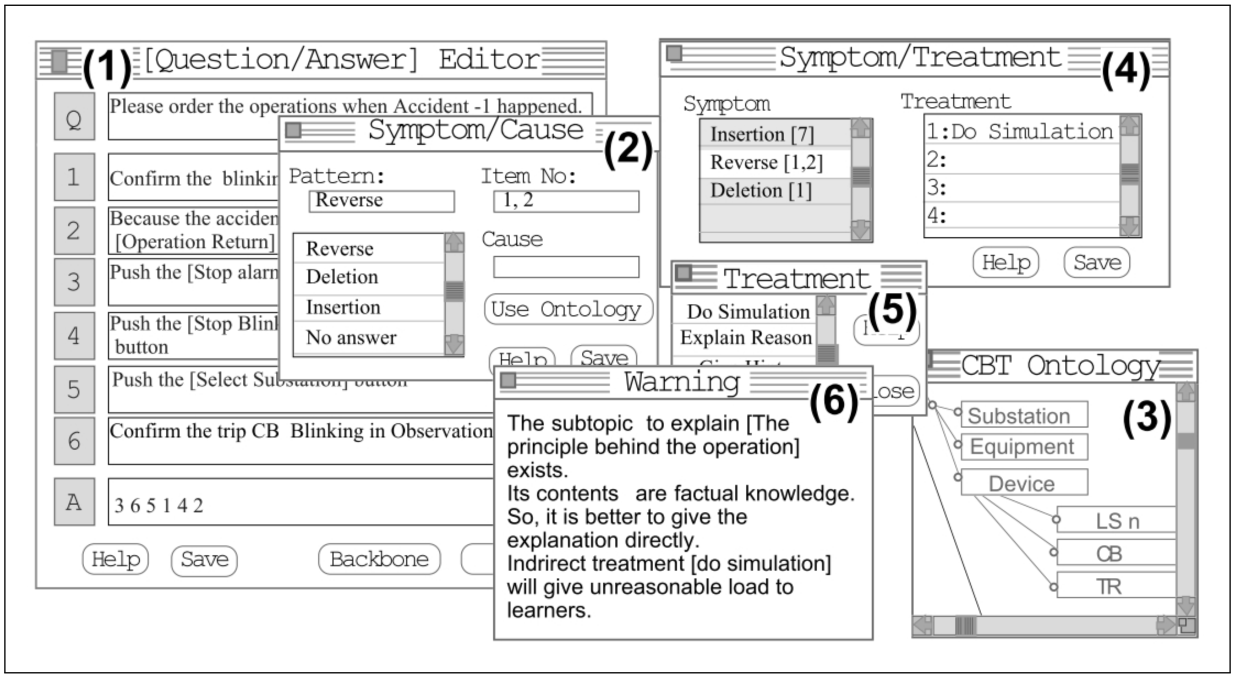

Semantic Support for Authoring Semantic Content
applicability
Guidelines for formalizing instruction
provide
Domain ontologies
Learning scenarios
papers
+ Rabbit to OWL: Ontology Authoring with a CNL-Based Tool [4]
+ Using Ontologies to Author Constraint-Based Intelligent Tutoring Systems [29]
supported by
Domain ontologies
papers
+ Inside Theory-Aware and Standards-Compliant Authoring System [19] + Task Ontology Makes It Easier To Use Authoring Tools [11] + Applications of a Collaborative Learning Ontology [2]
Proposed approach
Metadata validation
based on
Metadata schema as an ontology
Papers
+ Ontology-based Semantic Metadata Validation [24]
supported by
Task ontologies
Set of value restrictions
Papers
+ Automatic Metadata Generation using Associative Networks [25] + Hybrid system for generating learning object metadata [20]
+ Value-Added Metatagging: Ontology and Rule Based Methods for Smarter Metadata [8]
Metadata generation
provide
Suggestion of metadata values
Papers
+ Automatic Metadata Generation using Associative Networks [25] + Hybrid system for generating learning object metadata [20]
Creation of metadata values
papers
supported by
+ Taking advantage of metadata semantics: the case of learning-object-based lesson graphs [21] + Automatic Metadata Generation using Associative Networks [25] + Automatic Generation of Metadata for Learning Objects [26]
Learning objects metadata
Learning object discovery
based on
Set of suggestions and value restrictions
papers
+ Enhancing Educational-Material Retrieval Using Authored-Lesson Metadata [22]

Figure 3.1: Related work classification according to its applicability in the support for authoring content

3.1.1 Guidelines for formalizing instruction

Creation of learning content is a process which includes several stages. It is time consuming and error prone. That is why it is necessary to develop technologies and tools that make this process easier and more effective for the authors. In [11], the major concern is that authoring tools are difficult to develop, and usually appropriate support to the authors is missing. Mizoguchi et al. present an authoring tool based on task ontologies. The tool uses the ontology to serve as a guideline for the authors, where they create a learning scenario based on the structure defined in the task ontology, and the system gives feedback when a conflict occurs. Figure 3.2 shows the interface of the developed authoring tool. In this example, the author has chosen to create a question/answer of type *ordering question* from a list defined in the task ontology (not shown in the image). In window 1, the author inputs the question and answers. In window 2, the author defines the possible set of learner's errors, called 'symptom', along with its cause, selecting one from the domain knowledge represented in the ontology, as shown in window 3. The treatment of each symptom is specified in window 4, as a selection from a list (window 5). Finally, because the selected treatment violates a symptom/treatment constraint defined in the ontology, the tool displays a warning message (window 6) explaining the error and recommending a better treatment. We can see that the tool helps the authors to define the educational tasks using terms and vocabulary from the ontology (symptoms, treatments, causes) and that they have liberty to choose different elements to construct the learning scenario, but when the tool detects an inconsistency between the ontology and the generated model, it can provide a warning message to the authors.

Continuing with this line of research, Mizoguchi et al. in [2] present the collaborative learning ontology. The ontology provides conceptual knowledge in order to describe and evaluate collaborative learning scenarios. One of the applications of the ontology presented in the paper is an authoring tool for collaborative learning scenarios. The tool uses the ontology to assist the designers in the process of constructing the collaborative learning scenario. For example, when a designer selects a task, the tool suggests the most adequate



Q Please order the operations when Accident -1 happened.
1 Confirm the blinki
2 Because the acciden [Operation Return]
3 Push the [Stop alarn
4 Push the [Stop Blin button
5 Push the [Select Substation] button
6 Confirm the trip CB Blinking in Observation
A 3 6 5 1 4 2

Pattern: Reverse
Item No: 1, 2
Cause

Reverse
Deletion
Insertion
No answer

Symptom	Treatment
Insertion [7]	1:Do Simulation
Reverse [1,2]	2:
Deletion [1]	3:
	4:

Do Simulation
Explain Reason

The subtopic to explain [The principle behind the operation] exists.
Its contents are factual knowledge.
So, it is better to give the explanation directly.
Indrirect treatment [do simulation] will give unreasonable load to learners.

Figure 3.2: Interface of an Ontology based Authoring Tool (picture taken from [11])

way to achieve the learning goals associated with the task in the ontology.

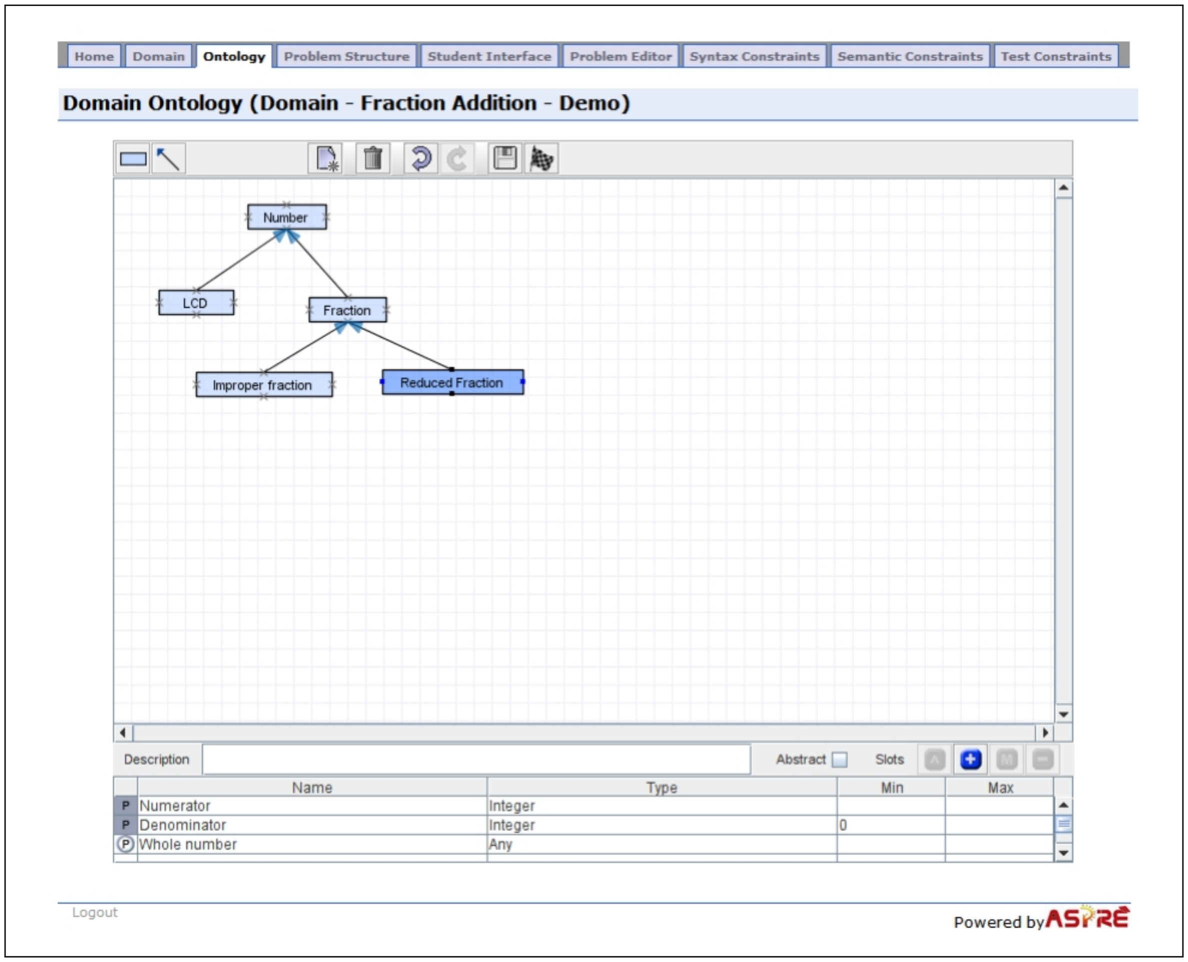
In a later work, Mizoguchi et al. [19] develop a fully aware authoring system which understands the different learning theories used in the generated content. The intelligent authoring system is built on an ontology that defines the necessary concepts for understanding instructional design. The tool involves two types of users: scenario authors and knowledge authors. First, a knowledge author describes instructional and learning strategies with the use of the ontology. This knowledge is stored in the system. Next, a scenario author can model a particular instructional process using the knowledge in the system, and because of such knowledge, the tool can provide guidelines for defining the model. This guideline consists of suggestions to the author depending on the learning theory being used, and not on basic built-in procedures. Another feature is the possibility to give scenario explanations to the authors, so they will be alerted when learning goals cannot be achieved due to a design problem in the current model. This approach clearly allows the authors to do a more accurate and structured authoring process, since they need to build the scenario step-by-step to fulfill all the learning goals specified in the underlining theory.

We have seen so far how an authoring tool can use an existing ontology to guide users in the authoring process of learning scenarios and tasks. Now, we will see tools that are designed to help the authors to create domain ontologies than can be used as learning content, or inputs for authoring tools. In [29], the creation of a domain ontology is the center of the authoring process, where the author defines concepts and their properties of the teaching domain in an ontology. That ontology will be used later to generate problems and answers in a constraint-based web tutor. ASPIRE is presented in Figure 3.3. The authoring environment guides the author through a series of steps that includes: selecting a teaching approach, develop a domain ontology with the relevant concepts to the task (current step shown in the figure), describe the structure of problems and solutions using the previously defined ontology, and provide a set of problems with solutions based on the solution structure. Finally, the system extracts syntactic and semantic constraints from the ontology and the sample problems/solutions, and delivers a complete running HTML tutoring system. It is important to highlight that the domain ontology in this case not only works to define the learning content, but also to construct the structure of the tutor.

In [4], an authoring tool to create a domain ontology using controlled natural language is presented. With ROO, novice authors without ontology engineering experience can model domain ontologies by describing the domain using structured English sentences. The tool provides suggestions and feedback to help the authors in the authoring process. When the authors finish describing the domain, ROO parses the sentences and provides error messages to the authors when there are potentially ambiguous sentences, that will help them to fix the problems. The system generates an OWL ontology when all the sentences are valid. The resulting domain ontology can be used later to develop learning content.

3.1.2 Metadata validation

This thesis focuses on learning objects metadata validation using semantic web technologies as part of the authoring process. Although correctness of metadata is essential for the



Domain Ontology (Domain - Fraction Addition - Demo)

	Name	Type	Min	Max
P	Numerator	Integer		
P	Denominator	Integer	0	
Ⓟ	Whole number	Any		

Figure 3.3: ASPIRE's Ontology workspace (picture taken from [29])

reusability of learning content and the correct learning of students, not so much work has been done in this direction. In [24], the authors want to unify different databases about museums in a single collection, but because the information comes from different sources the data needs to be validated and combined at the semantic level. The proposed process for harmonization and validation of the data is as follows:

1. The data from the different databases are exported to a common XML representation and validated against an XML Schema.

2. The XML records are transformed into RDF tuples.

3. The RDF-statements are validated against a defined RDF Schema.

It is necessary to highlight some details. Because of the use of different XML sources and the properties of RDF, the mapping of terms from XML to RDF tuples can sometimes be ambiguous, and human intervention is necessary. Additionally, in the semantic validation step, only two kinds of constraints can be checked: *domain*, and *range*. These two limitations reduce the impact that the application can have in other contexts, besides the one presented.

We will see in later chapters of this thesis how by using more powerful technologies (XSLT and OWL2) and a similar approach, the quality of the semantic validation can be improved. By having the option to specify more constraints and explain to authors what is causing problems in the content, a more exhaustive correctness of the metadata semantics can be accomplish than in [24].

3.1.3 Metadata generation

At this point, we have seen how ontologies are the central component to help with the creation of learning content, but now we will analyze what work has been done where the central focus is the learning object metadata.

Continuing in the line of metadata values in the learning content, not only validation can be done, but also automatic creation or suggestion of metadata. The authors of [8] present a system to improve the productivity of metadata authoring, and the quality of such metadata. The paper focuses on suggestion of metadata values for learning resources that belong together in assemblies (collections of relatively independent units) and repositories, where the feature of 'similar' objects can be exploited. Four methods are presented: inheritance, accumulation, content similarity and semantical similarity measure, which uses ontology matching to find similar metadata records. The figure 3.4 shows the process in the semantical similarity measure: (a) existing metadata values are mapped to ontology concepts, (b) which trigger inference rules that operate on the vocabularies derived from the ontology, (c) these rules find a set of semantically similar records, (d) from which a set of suggested values for the missing elements of the current learning object metadata is extracted and presented to the authors. Although the authors still need to fill up all the metadata elements, the list of suggested values decreases the amount of work and if the learning objects in the repository have high quality metadata, the suggested values will preserve that quality.

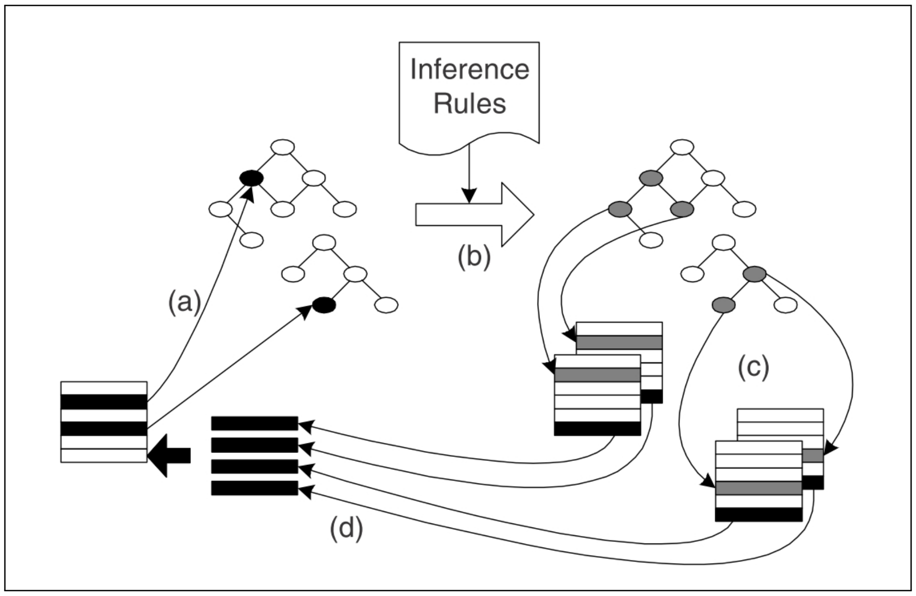


Figure 3.4: Generation of suggested metadata values through semantical similarity (picture taken from [8])

Automatic generation of metadata for learning objects is presented in [26], where the authors propose a probabilistic model and the use of ontologies to classify learning objects and associate learning metadata to them. The first step in the approach is the development of an ontology that describes the domain knowledge, and where its concepts represent categories to classify the learning objects. In other words, the ontology represents a taxonomy in which elements can be categorized. After the ontology is complete, each node or category is labeled with descriptive keywords. Finally, the system classifies the learning objects using the ontology and associates to them the corresponding metadata. Collections of learning objects that are mainly just text documents, and without previously created metadata can benefit from this technique.

A hybrid system that mixes automatic and manual generation of LOM is presented in [20]. The authors consider the learning scenario or lesson as a graph, in which each node is a learning object, and the links define relations between them. Based on the graph notion, the tool can perform LOM generation and LOM validation. Generation of metadata corresponds to technical elements like *size* or *language*, where the values are extracted from the context and automatically generated. For the metadata that is related to educational semantics like *density* or *typical learning time*, the system uses the semantics in the learning objects graph to generate potential values (suggestions) and value boundaries (restrictions) for the metadata. These suggestions and restrictions are shown to the author to support the manual instantiation of the metadata. The LOM validation uses the generated restrictions (from the semantics of the graph) to mark every metadata element as *invalid* (deduced restrictions do not hold), *undefined* (empty metadata elements), or *not invalid* (completeness and correctness are satisfied). This dual approach helps the authors to produce more accurate and high quality metadata, and at the same time if the quality of the learning objects is high, the system can generate better suggestions and restrictions for new content.

The idea for the suggestion of metadata is extended and improved in [21]. The paper introduces a framework to handle the metadata that is missing, incorrect, or incomplete in learning object repositories by using rules and a diffusion mechanism. The framework works by applying influence rules to the learning objects graph to generate a set of contextualized or inferred metadata values (CMV's). Afterwards, the influence rules are applied to both the generated CMV's and the metadata values of the content generating a new set of CVM's. The process is repeated iteratively until a CVM's converges. The framework allows the use of different metadata processing strategies, for example, by using probabilistic analysis or rules to generate restrictions. Depending on the strategy LOM validation can be accomplished, as well. Figure 3.5 shows an example of LessonMapper2, a tool developed using the proposed framework. The *semantic density* metadata values of the learning objects L1, L3 and L6 are shown. The author asks for suggestions and restrictions of the value for L1. The system uses the metadata diffusion and infers that the semantic density of L1 should be less or equal to the semantic density of L3. In addition, it suggests possible values for the element: medium density (from the value of L6), low density (from the value of L3), and very high density (from others objects in the repository). Furthermore, suggested values that satisfy and not the inferred restrictions have different colors.

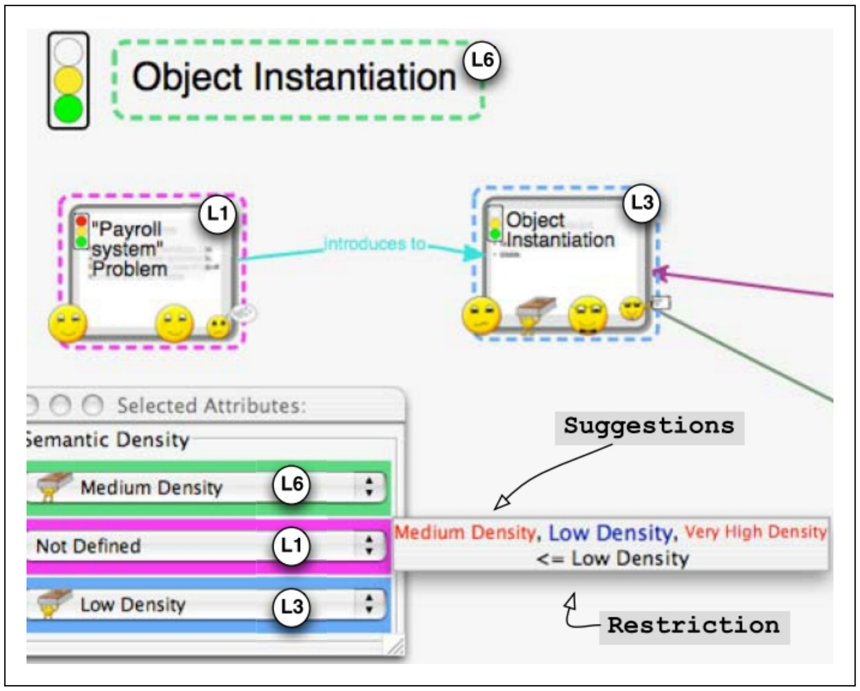


Figure 3.5: Suggestions and restrictions of metadata in LessonMapper2 (picture taken from [21])

Continuing in the scenario where a repository contains learning objects with different degrees of metadata quality, [25] describes a propagation model using associative networks. An associative network connects resources according to a measure of similarity. After the creation of the associative network, the metadata is propagated, and when a resource is missing a metadata value, the element is set to the value from the metadata-rich resource. Human validation is necessary because the system only propagates metadata values and does not take into account properties of the learning objects. Furthermore, the initial quality of the original resources with set metadata values will determine the quality of the extrapolated values in other learning objects.

3.1.4 Learning object discovery

Efficient and accurate retrieval of learning objects in repositories let them be sharable and reusable. In [22], the authors extend the typical keyword searching of resources, by using the desired purpose of the learning object as a search parameter. For example, if an author looks for an example for a mathematical concept, the tool will suggest learning objects that have been used before as examples for such concept. The retrieval engine is part of an authoring tool that uses the concept of graphs to represent the learning objects as part of a lesson or learning scenario. When the author adds a new empty node to the current graph, the retrieval engine creates a series of values suggestions and restrictions from the graph consistency analyses and uses them as classifiers to find the most adequate learning object. The results are combined with a keyword search, and then presented to the author. Although the tool represents the relationships between the learning objects using a graph, ontologies can be used with the same approach, because they allow to model complex relations and constraints between the learning objects.

3.1.5 Summary

We have seen how Semantic Web technologies are used in different tools and frameworks to support the creation and validation of learning content. If we think of those applications as a whole, and not as isolated tools, we can catalog them according to their role in the stages of the authoring process. Figure 3.6 shows a diagram using this classification.

In the pre-authoring stage, conceptual knowledge is created in the form of ontologies to support pedagogical or domain model creation. In [11, 19], ontologies to represent instructional design are built to support the creation of learning scenarios. In a similar way, [2] presents an ontology that describes collaborative learning scenarios. [4, 29] describe tools that facilitate the creation of domain ontologies. In this stage, the main user is the knowledge author, who was a deep understanding of the target domain.

A scenario author works in the authoring stage to create learning objects and its metadata, and learning scenarios. [8, 20] present tools that suggest metadata values to the authors when they are authoring learning resources. In [11, 19, 2], the author is guided to the process of constructing learning scenarios, either by offering recommendations of which resources to use, or by receiving warning messages when the scenario does not accomplish the desired learning goals. Additionally, [22] describes a tool that helps the author to find and select the most adequate learning objects to use in the learning scenario that is being authored.

Finally, automated validation can be performed to improve the quality of the generated content. Metadata validation as presented in [24] uses an ontology as a schema to check the correctness of the metadata. The approach proposed in this thesis also focuses on learning objects metadata validation using ontologies. Another way to check the metadata is using value restrictions generated from the semantics of the learning objects, as presented in [20]. For repositories of learning objects with missing, incomplete, or poor quality metadata, different techniques have been proposed. In [21], influence rules generate a set of inferred metadata values used to update the metadata values in the learning objects. Propagation from high quality to poor quality metadata resources is described in [25], and [26] presents a probabilistic method to assign predefined metadata to learning objects.

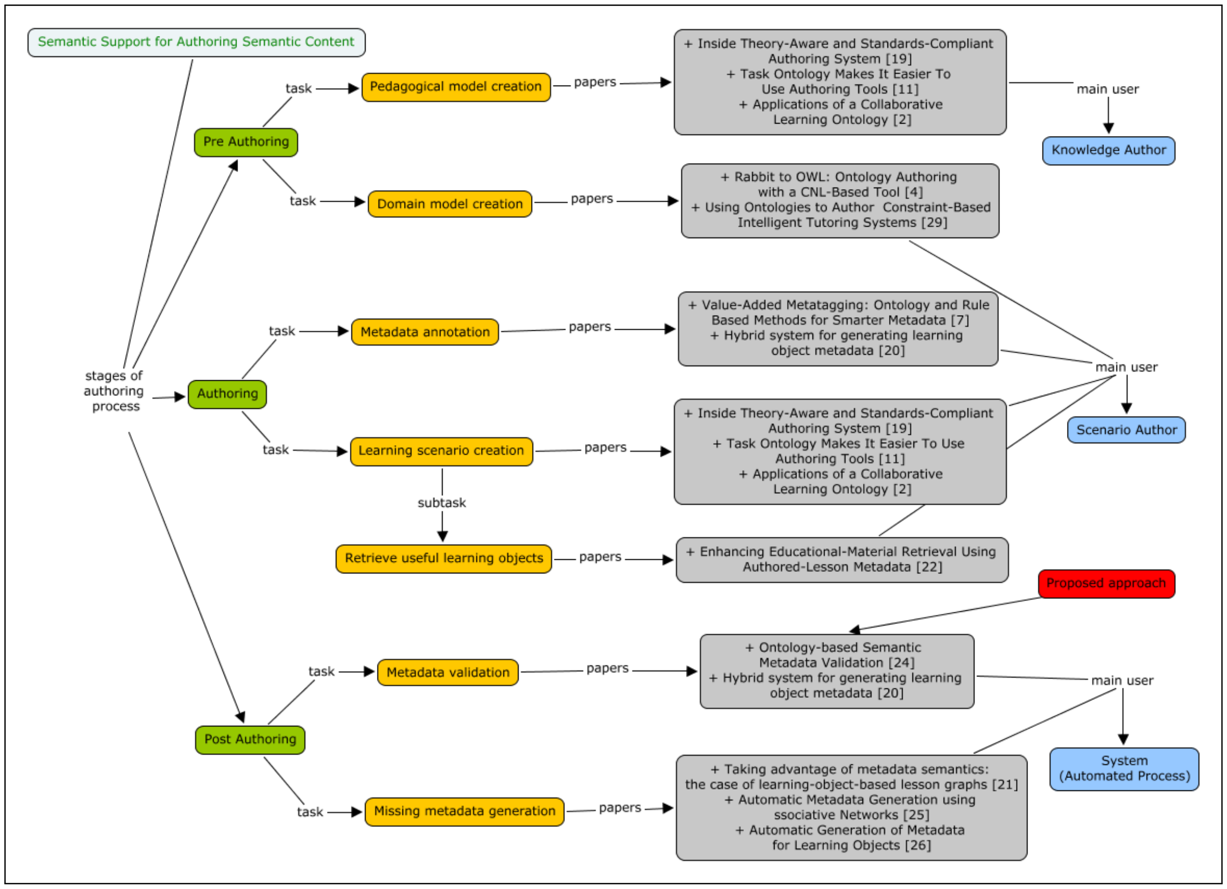

Semantic Support for Authoring Semantic Content
stages of authoring process
Pre Authoring
Authoring
Post Authoring
task
Pedagogical model creation
Domain model creation
Metadata annotation
Learning scenario creation
subtask
Retrieve useful learning objects
Metadata validation
Missing metadata generation
papers
+ Inside Theory-Aware and Standards-Compliant Authoring System [19]
+ Task Ontology Makes It Easier To Use Authoring Tools [11]
+ Applications of a Collaborative Learning Ontology [2]
+ Rabbit to OWL: Ontology Authoring with a CNL-Based Tool [4]
+ Using Ontologies to Author Constraint-Based Intelligent Tutoring Systems [29]
+ Value-Added Metatagging: Ontology and Rule Based Methods for Smarter Metadata [7]
+ Hybrid system for generating learning object metadata [20]
+ Inside Theory-Aware and Standards-Compliant Authoring System [19]
+ Task Ontology Makes It Easier To Use Authoring Tools [11]
+ Applications of a Collaborative Learning Ontology [2]
+ Enhancing Educational-Material Retrieval Using Authored-Lesson Metadata [22]
+ Ontology-based Semantic Metadata Validation [24]
+ Hybrid system for generating learning object metadata [20]
+ Taking advantage of metadata semantics: the case of learning-object-based lesson graphs [21]
+ Automatic Metadata Generation using ssociative Networks [25]
+ Automatic Generation of Metadata for Learning Objects [26]
Proposed approach
main user
Knowledge Author
Scenario Author
System (Automated Process)

Figure 3.6: Related work classification according to its role in the authoring process

3.2 Background: Gap Detection in ActiveMath

This thesis is motivated from a previous work done to automatize the detection and management of gaps in ActiveMath. The Gap Detection Tool presented in [12] represents an effort to ensure the high quality of the content in ActiveMath, using exhaustive verification of the content metadata.

3.2.1 Types of gaps

Existing gaps in the learning content of ActiveMath have been divided into three categories:

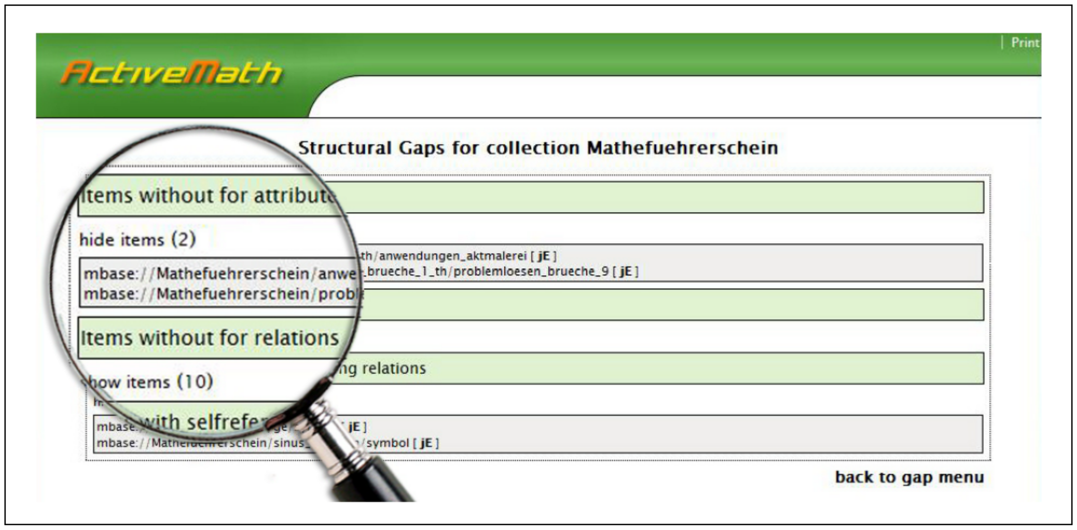
Structural gaps Learning objects missing metadata connecting to other resources, or with incorrect relational metadata. There are many relations that can be applied between learning objects, but the problem with these relations is that errors are hard to identify because they are directly related to the semantics of the relation.

Linguistic gaps Multilingual learning content where translations are not updated when the learning content changes. After a collection is translated, new added items usually are not translated into all target languages, or learning objects are modified only in one language.

Didactic gaps Missing learning objects for all possible (or, at least, typical) learning situations. When the system needs a specific learning object, and it is not available, the student can get one that is, for example, too hard or too easy.

3.2.2 Gap detection

The implemented tool can detect gaps belonging to all categories. For structural gaps, when a learning object misses a required relation (for example, definitions not connected to symbols, or proofs without a link to theorems or lemmas), a gap is detected. Verification of semantic correctness of metadata relations happens when the tool detects that a learning object is related to itself, or when it is related to a non existing object. Figure 3.7 demonstrates the results of structural gap detection presented to an author. Inconsistencies occurred because several learning objects are not connected to target concepts (missing the *for* relation).



Structural Gaps for collection Mathefuehrerschein

Items without for attribute

hide items (2)

mbase://Mathefuehrerschein/anwe th/anwendungen_aktmalerei [jE]
mbase://Mathefuehrerschein/probl brueche_1_th/problemloesen_brueche_9 [jE]

Items without for relations

how items (10)

ng relations

with selfrefe

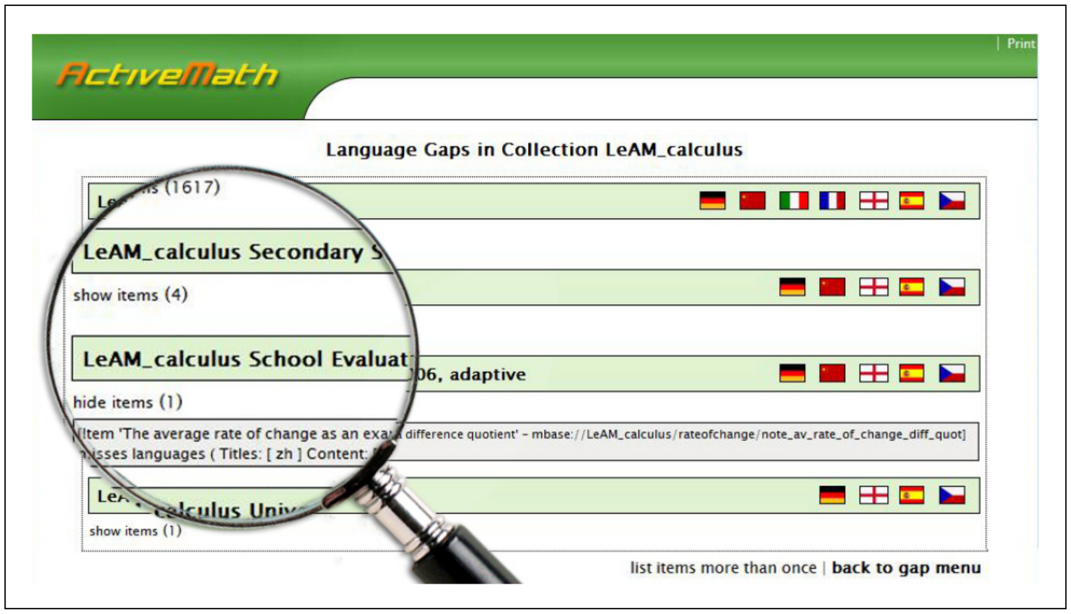
mbase: [jE]
mbase://Mathefuehrerschein/sinus /symbol [jE]

back to gap menu

Figure 3.7: Structural gap detection (picture taken from [12])

Linguistic gap detection validates that each learning object appears only in linguistically valid contexts and that metadata is also translated along with the content translations. For example, a learning object having its content translated into Spanish should also have a title in Spanish. Figure 3.8 demonstrates the results of linguistic gap detection performed for three multilingual ActiveMath courses.

For detection of the didactic gaps, the tool uses typical user models, base on the information about the students taken from the course, to generate gaps about learning objects with non-existing characteristics for the possible learning scenarios. Figure 3.9 demonstrates an example of a didactic gap observed in ActiveMath content; a missing exercise with particular characteristics has been found.



Language Gaps in Collection LeAM_calculus

Le... ...s (1617)

LeAM_calculus Secondary S

show items (4)

LeAM_calculus School Evaluat ...06, adaptive

hide items (1)

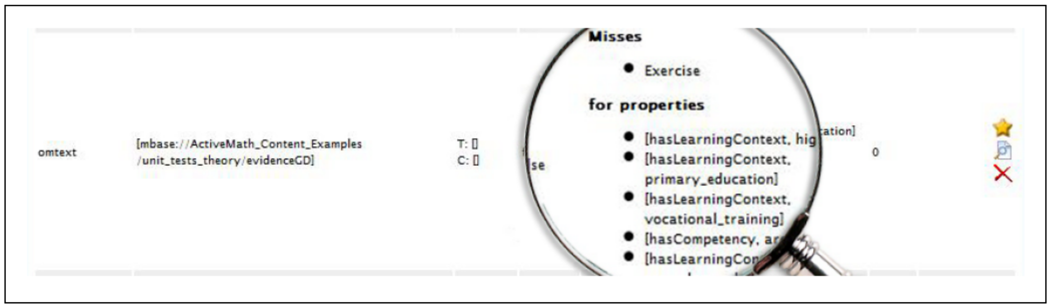
[Item 'The average rate of change as an exa... difference quotient' - mbase://LeAM_calculus/rateofchange/note_av_rate_of_change_diff_quot]
...sses languages (Titles: [zh] Content:

LeA... calculus Univ...

show items (1)

list items more than once | back to gap menu

Figure 3.8: Linguistic gap detection (picture taken from [12])



omtext

[mbase://ActiveMath_Content_Examples /unit_tests_theory/evidenceGD]

T: ▯
C: ▯

se

Misses

- Exercise

for properties

- [hasLearningContext, hig
- [hasLearningContext, primary_education]
- [hasLearningContext, vocational_training]
- [hasCompetency, ar
- [hasLearningCo

ation]

0

Figure 3.9: Didactic gap detection (picture taken from [12])

3.2.3 Limitations

Even when the system is capable to detect all three types gaps, only two types of gaps about the semantic correctness of metadata can be detected, which limits the verification of the metadata. As explained in chapter 2, section 2.3, the relations and elements in pedagogical and semantic metadata define many semantic relations between the learning objects, that cannot be detected and have a high impact in the quality of the learning content.

Because of this problem with the semantics in the structural gap detection, a new tool needed to be developed. This thesis focuses on how to develop a complete semantic gap detection tool that can detect problems with the semantics of the relations between learning objects.

Chapter 4

Approach

This chapter presents a mechanism for semantic gap detection, which validates learning object metadata against a formal definition of metadata requirements. This validation is necessary because collections of learning objects can have poor-quality metadata, which can cause severe errors in the system. This poor-quality metadata is the result of the loose control in metadata authoring and the difficulties of this task. This thesis proposes to solve this problem by formalizing the requirements of the metadata usage, and then verifying the metadata against such requirements definition.

The approach is composed of two stages. The initial one is to formalize the metadata requirements using an OWL2 ontology as metadata schema. The second stage is an automated process that validates the metadata against the previously defined metadata schema and generates human-readable explanations for the gaps.

4.1 Metadata Requirements Definition

Metadata requirements are necessary to serve as a guideline to check if the metadata in the learning objects is being used properly. In the proposed approach, these requirements are formalized as an OWL2 ontology. OWL2 allows to specify hierarchies of classes to represent the network of learning objects and the metadata elements, and instances or individuals of such classes to represent the learning objects and the values of those metadata elements. To describe how learning objects relate to each other and to assign metadata to the learning objects, OWL2 object properties are used. Object properties are relations which link elements in the ontology, and can have restrictions that define the semantics and usage of those relations. Among the most common restrictions are: domain (which elements can use a relation), range (to which elements a relation can connect to), and disjoint (define classes that are mutually exclusive). A complete reference of restrictions and all properties in OWL2 can be found in [7]. For example, the *'example_addition'* learning object is an instance of the *'Example'* class. The *'mathematics'* value is an instance of the *'Field'* class. To indicate that the *'example_addition'* is of the field of mathematics, the *'hasField'* object property is used. This relation has a range restriction which indicates that it can only link to values of the *'Field'* class.

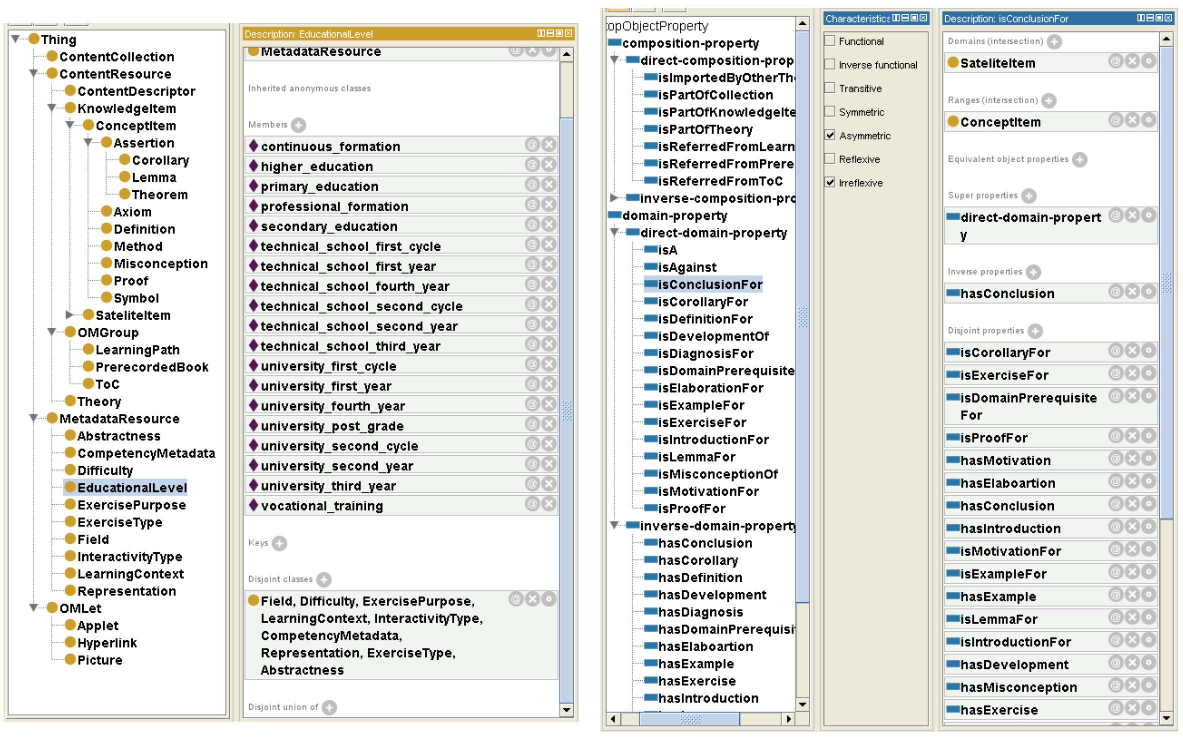
A complete metadata requirements ontology was created to formalize the metadata usage in ActiveMath. This ontology contains all the descriptive, pedagogic and semantic metadata that is used in the learning objects. Figure 4.1a shows on the left side the classes in the ontology. These classes represent the metadata elements. On the right side, the values for the *EducationalLevel* class are displayed. Figure 4.1b shows in the left side the object properties in the ontology, which represent the relations among the metadata. On the right side, the restrictions for the *isConclusionFor* relation are shown. Protégé[1] was used to create and display the ontology.

In the rest of the thesis, the ontology containing all the requirements for the metadata is called *metadata schema*.

4.2 Metadata Validation

After the metadata schema has been created, the metadata validation process can be fully automated. This process consists of four steps. The first one is to convert the metadata of ActiveMath learning objects represented in OMDoc/XML to OWL2 ontologies. Then, detection of metadata gaps is done by using a semantic reasoner to discover formal conflicts that appear in the OWL2 ontologies. Third, the OWL2 axioms corresponding to metadata that are responsible for the gaps are identified. Finally, human-readable explanations for

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



Thing
ContentCollection
ContentResource
ContentDescriptor
KnowledgeItem
ConceptItem
Assertion
Corollary
Lemma
Theorem
Axiom
Definition
Method
Misconception
Proof
Symbol
SatelliteItem
OMGroup
LearningPath
PrerecordedBook
ToC
Theory
MetadataResource
Abstractness
CompetencyMetadata
Difficulty
EducationalLevel
ExercisePurpose
ExerciseType
Field
InteractivityType
LearningContext
Representation
OMLet
Applet
Hyperlink
Picture

Description: EducationalLevel
MetadataResource
Inherited anonymous classes
Members
continuous_formation
higher_education
primary_education
professional_formation
secondary_education
technical_school_first_cycle
technical_school_first_year
technical_school_fourth_year
technical_school_second_cycle
technical_school_second_year
technical_school_third_year
university_first_cycle
university_first_year
university_fourth_year
university_post_grade
university_second_cycle
university_second_year
university_third_year
vocational_training
Keys
Disjoint classes
Field, Difficulty, ExercisePurpose, LearningContext, InteractivityType, CompetencyMetadata, Representation, ExerciseType, Abstractness
Disjoint union of

(a) Metadata elements and their values

topObjectProperty
composition-property
direct-composition-prop
isImportedByOtherTh
isPartOfCollection
isPartOfKnowledgeIte
isPartOfTheory
isReferredFromLearn
isReferredFromPrere
isReferredFromToC
inverse-composition-pro
domain-property
direct-domain-property
isA
isAgainst
isConclusionFor
isCorollaryFor
isDefinitionFor
isDevelopmentOf
isDiagnosisFor
isDomainPrerequisite
isElaborationFor
isExampleFor
isExerciseFor
isIntroductionFor
isLemmaFor
isMisconceptionOf
isMotivationFor
isProofFor
inverse-domain-property
hasConclusion
hasCorollary
hasDefinition
hasDevelopment
hasDiagnosis
hasDomainPrerequisi
hasElaboartion
hasExample
hasExercise
hasIntroduction

Characteristics
☐ Functional
☐ Inverse functional
☐ Transitive
☐ Symmetric
☑ Asymmetric
☐ Reflexive
☑ Irreflexive

Description: isConclusionFor
Domains (intersection)
SatelliteItem
Ranges (intersection)
ConceptItem
Equivalent object properties
Super properties
direct-domain-property
Inverse properties
hasConclusion
Disjoint properties
isCorollaryFor
isExerciseFor
isDomainPrerequisiteFor
isProofFor
hasMotivation
hasElaboartion
hasConclusion
hasIntroduction
isMotivationFor
isExampleFor
hasExample
isLemmaFor
isIntroductionFor
hasDevelopment
hasMisconception
hasExercise

(b) Metadata relations and their restrictions

Figure 4.1: ActiveMath metadata schema

the gaps are generated. Figure 4.2 summarizes the process.

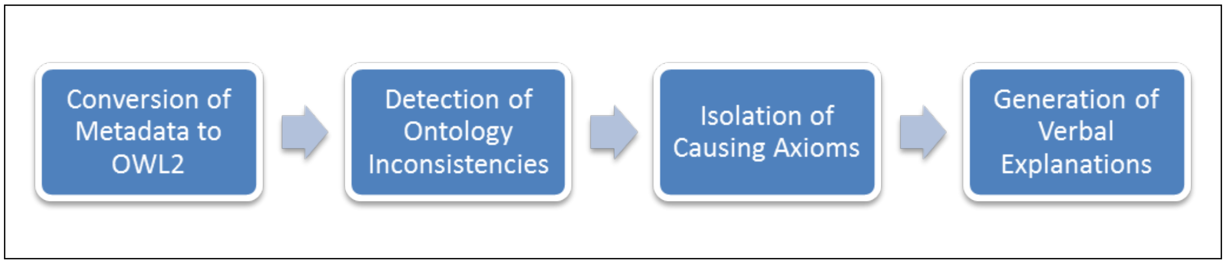

Conversion of Metadata to OWL2
Detection of Ontology Inconsistencies
Isolation of Causing Axioms
Generation of Verbal Explanations

Figure 4.2: Semantic gap detection procedure

4.2.1 Conversion of metadata to OWL2

The purpose of metadata and metadata standards is to provide a machine-readable description for information items. In this way, they become easily discoverable based on their properties. Most of the metadata standards, such as Dublin Core[2], Learning Object Metadata[3], and IMS Learning Design[4] use XML-based languages. XML serves the main purpose of enabling automatic discoverability, but unfortunately it provides little support for verification. This means that the first step required for the metadata validation is to transform all the metadata into OWL2.

ActiveMath, as mentioned earlier, uses OMDoc for knowledge representation, both the content and its metadata. OMDoc is an XML language, and OWL2 is usually also represented using XML serialization [6], therefore, in general any XML transformation language can be used to transform OMDoc into OWL2. XSLT was chosen as transformation language because is one of the best known and a W3C recommendation.

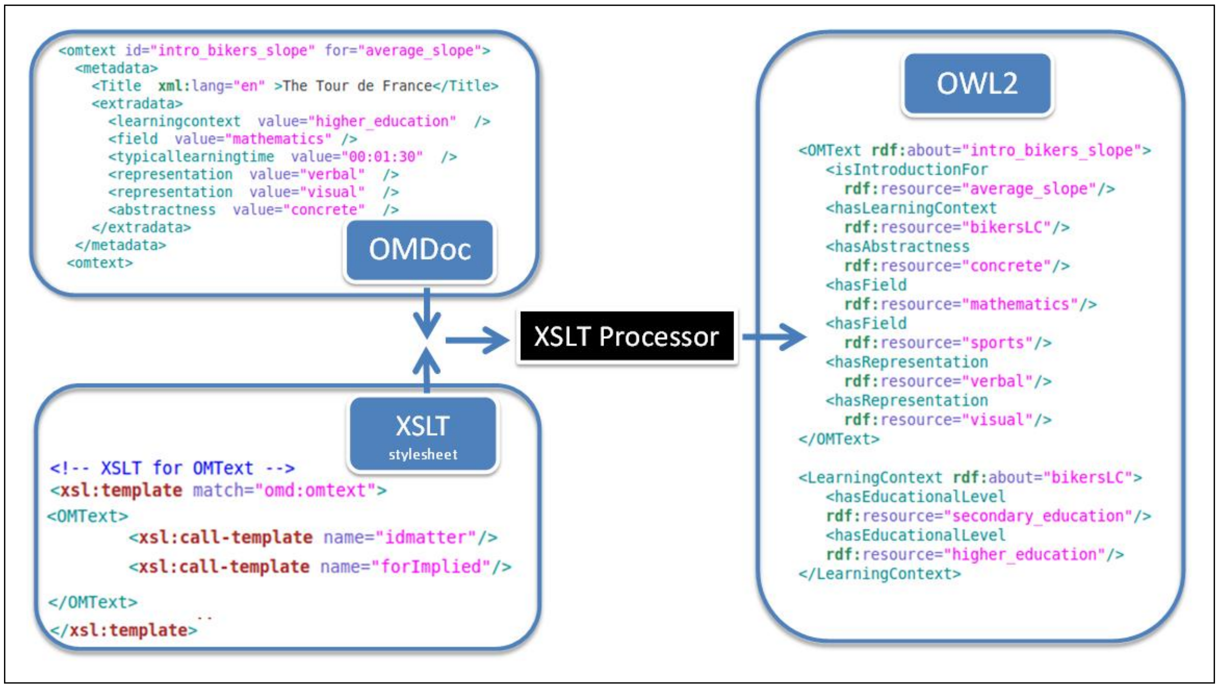
The first subtask to use XSLT is to create an XSLT stylesheet, which contains all the template rules to produce the desired output document. A stylesheet document was created manually, based on the OMDoc DTD documents and the metadata schema, containing the rules to convert from OMDoc to OWL2. One of those rules is an import declaration to link the current ontology to the metadata schema. In this way, the semantic reasoner knows that the current ontology representing the learning object metadata needs to be validated against the specified metadata schema, which allows to check for inconsistencies (gaps) in steps two and three. If the structure of OMDoc, or the metadata schema changes, only an update in the XSLT stylesheet is required; hence the rest of the process is not affected.

Once the XSLT stylesheet is ready, an XSLT processor applies the rules to the input document (the OMDoc file), and produces the output document (the OWL2 file). This process is applied to all the learning objects in a learning collection in OMDoc, producing a matching metadata collection in OWL2 format. It is important to explain that in ActiveMath, a learning object collection contains several OMDoc files, which at the same time have various learning objects. This step transforms file by file, but in the next steps, the analysis is focused in each individual learning object. In the rest of the thesis, the

[2]http://dublincore.org/documents/dc-xml/
[3]http://ltsc.ieee.org/wg12/
[4]http://www.imsglobal.org/learningdesign/ldv1p0/imsld_bindv1p0.html

resulting OWL2 models of learning objects and their metadata from this step are called *learning object ontologies*.

Figure 4.3 presents an example of XSLT transformation. The XSLT processor takes an OMDoc document containing a learning object of type *OMText*, and the XSLT stylesheet with the corresponding template rules. The output is an OWL2 ontology representing the metadata for the *'intro_bikers_slope'* learning object.



```
<omtext id="intro_bikers_slope" for="average_slope">
  <metadata>
    <Title  xml:lang="en" >The Tour de France</Title>
    <extradata>
      <learningcontext  value="higher_education"  />
      <field  value="mathematics" />
      <typicallearningtime  value="00:01:30"  />
      <representation  value="verbal"  />
      <representation  value="visual"  />
      <abstractness  value="concrete"  />
    </extradata>
  </metadata>
 <omtext>
```

```
<!-- XSLT for OMText -->
<xsl:template match="omd:omtext">
<OMText>
        <xsl:call-template name="idmatter"/>
        <xsl:call-template name="forImplied"/>
</OMText>
</xsl:template>
```

```
<OMText rdf:about="intro_bikers_slope">
   <isIntroductionFor
     rdf:resource="average_slope"/>
   <hasLearningContext
     rdf:resource="bikersLC"/>
   <hasAbstractness
     rdf:resource="concrete"/>
   <hasField
     rdf:resource="mathematics"/>
   <hasField
     rdf:resource="sports"/>
   <hasRepresentation
     rdf:resource="verbal"/>
   <hasRepresentation
     rdf:resource="visual"/>
</OMText>

<LearningContext rdf:about="bikersLC">
   <hasEducationalLevel
   rdf:resource="secondary_education"/>
   <hasEducationalLevel
   rdf:resource="higher_education"/>
</LearningContext>
```

Figure 4.3: Example of XSLT transformation

4.2.2 Detection of ontology inconsistencies

Next step is to detect if there are metadata gaps in the collection. In terms of OWL2, this step is equivalent to formal inference of logical conflicts in each of the learning object ontologies in the collection. As mentioned before, each ontology imports the metadata schema, which ensures that all the metadata requirements must be fulfilled in order for each learning object ontology to be consistent. Consequently, all the metadata to be correct.

A semantic reasoner is used to detect whether an ontology is consistent or not. The consistency check requires little computation time, usually milliseconds, and even for larger ontologies - tens of thousands of axioms - the check can be done within a couple of seconds, as shown in [10].

Each learning object ontology in the collection is checked for inconsistencies. If the ontology is consistent, there are no metadata gaps in that ontology. On the other hand, an inconsistent ontology indicates semantic gaps, and it is necessary to identify their source. This is done in the next step. For checking the metadata of existing collections, this step is done at the learning object ontology level, where several learning objects can be analyze at the same time. However, the consistency check can be also done for a individual learning objects (see Chapter 5, Section 5.3), which is useful if the ontology contains a lot of learning objects or to verify the correctness of the learning object when the author is creating it.

Figure 4.4 shows an example of such inconsistency. The metadata relation *hasDomain-Prerequisite* indicates that the learning object has to be mastered before the current object. A defined restriction of type range, requires that the relation must only link to elements of type *ConceptItem* and not *Misconception*. An inconsistency occurs because the element *'intro_bikers_slope'* is linked with the relation *hasDomainPrerequisite* to the element *'ex_tour_de_fr'*, but the later is of type *Example*, which is not a subclass of *ConceptItem*, but of *SateliteItem*. This violates the defined range requirements of the relation.

4.2.3 Isolation of causing axioms

After inconsistencies have been detected in a learning object ontology, it is necessary to find the exact cause of the gaps responsible for them. In terms of OWL2 ontologies, the minimal subsets of axioms from the ontology that cause it to be inconsistent are called justifications. These justifications represent metadata gaps in the collection.

A semantic reasoner is necessary for this task, but not enough. Some reasoners (Pallet[5], HermiT[6]) can find, for consistent ontologies, justifications for entailments (inferences) and

[5]`http://clarkparsia.com/pellet/`
[6]`http://www.hermit-reasoner.com/`

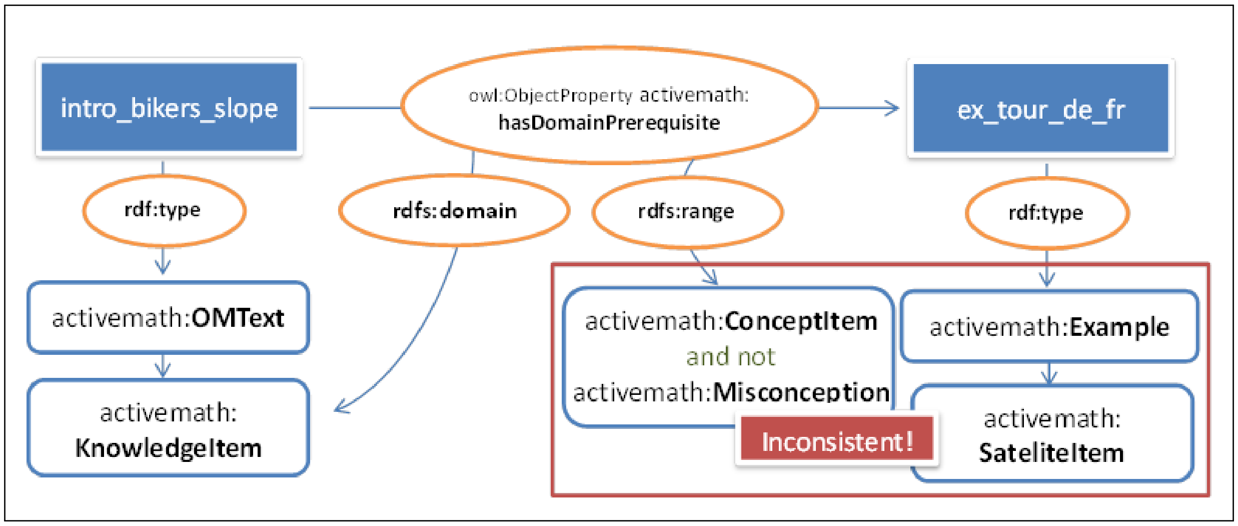
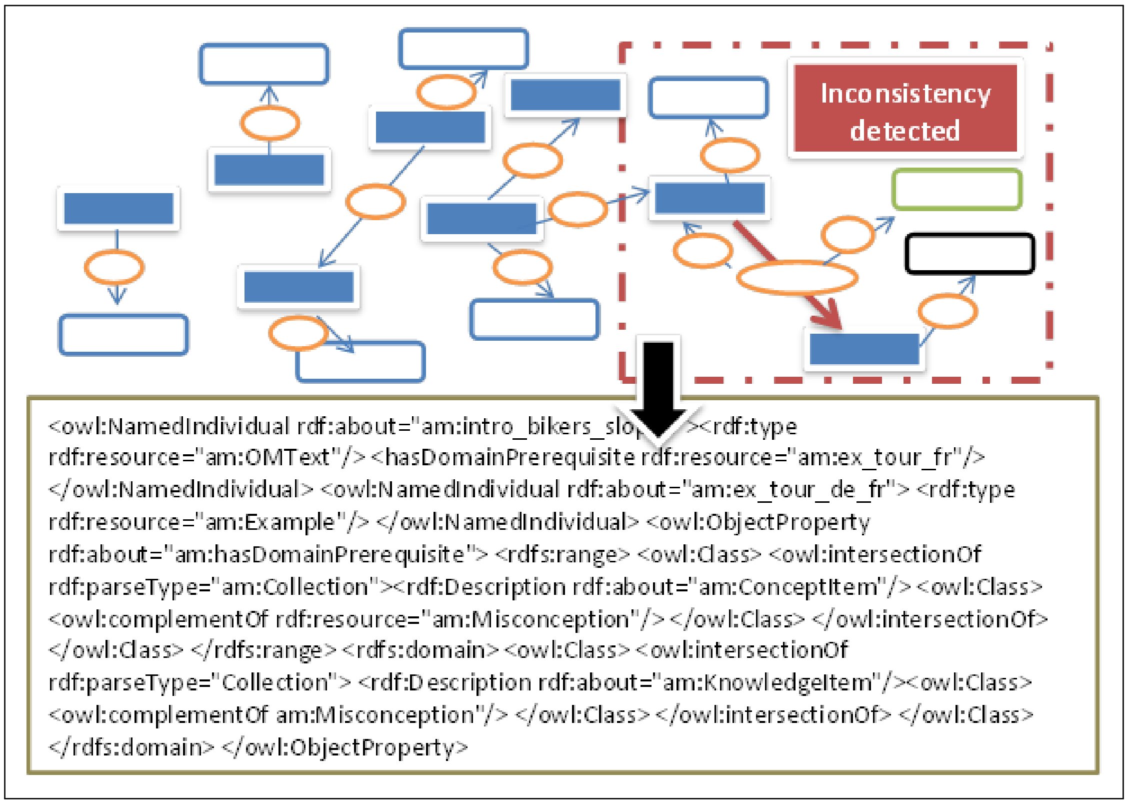


Figure 4.4: Inconsistency example. A range restriction is violated

unsatisfiable concepts (concepts which cannot have any individuals), but not justifications for inconsistencies. An external algorithm is necessary to work with the reasoner to discover the justifications for inconsistencies. As described in [10], the OWL API provides a black-box algorithm capable of finding justifications for inconsistent ontologies. This algorithm uses a two phase expand-contract procedure. The simple expansion phase creates a set containing all the axioms in the ontology, followed by a "divide and conquer" contraction phase. This phase finds a subset of axioms that make the ontology be inconsistent, and then prunes the subset until it is a minimal set of axioms that makes the ontology inconsistent, i.e. until it is a justification. The algorithm for computing all justifications is a hybrid of the algorithm for computing single justifications taken from [1] and combined with the algorithm for computing all justifications (Reiter's Hitting Set Algorithm) taken from [13]. The main disadvantage of this algorithm is the runtime performance: exponential with the number of justifications (computing all justifications is an inherently difficult problem).

This combination of reasoner and algorithm for justifications is applied to all the inconsistent metadata ontologies to generate the justifications, each one being a set of axioms that will be used to generate a human-readable verbal explanations in the next step.

Figure 4.5 shows how the axioms causing the inconsistency are isolated for the conflict represented by Figure 4.4. First, it is necessary to determine which property or metadata element is causing the inconsistency, then look to the other metadata of the learning object and compute the causing axioms from there. The resulting set of axioms can be quite complex. In order for the human author to understand it, the next step of the approach generates verbal summaries of the observed inconsistencies.



```
<owl:NamedIndividual rdf:about="am:intro_bikers_slo   ><rdf:type
rdf:resource="am:OMText"/> <hasDomainPrerequisite rdf:resource="am:ex_tour_fr"/>
</owl:NamedIndividual> <owl:NamedIndividual rdf:about="am:ex_tour_de_fr"> <rdf:type
rdf:resource="am:Example"/> </owl:NamedIndividual> <owl:ObjectProperty
rdf:about="am:hasDomainPrerequisite"> <rdfs:range> <owl:Class> <owl:intersectionOf
rdf:parseType="am:Collection"><rdf:Description rdf:about="am:ConceptItem"/> <owl:Class>
<owl:complementOf rdf:resource="am:Misconception"/> </owl:Class> </owl:intersectionOf>
</owl:Class> </rdfs:range> <rdfs:domain> <owl:Class> <owl:intersectionOf
rdf:parseType="Collection"> <rdf:Description rdf:about="am:KnowledgeItem"/><owl:Class>
<owl:complementOf am:Misconception"/> </owl:Class> </owl:intersectionOf> </owl:Class>
</rdfs:domain> </owl:ObjectProperty>
```

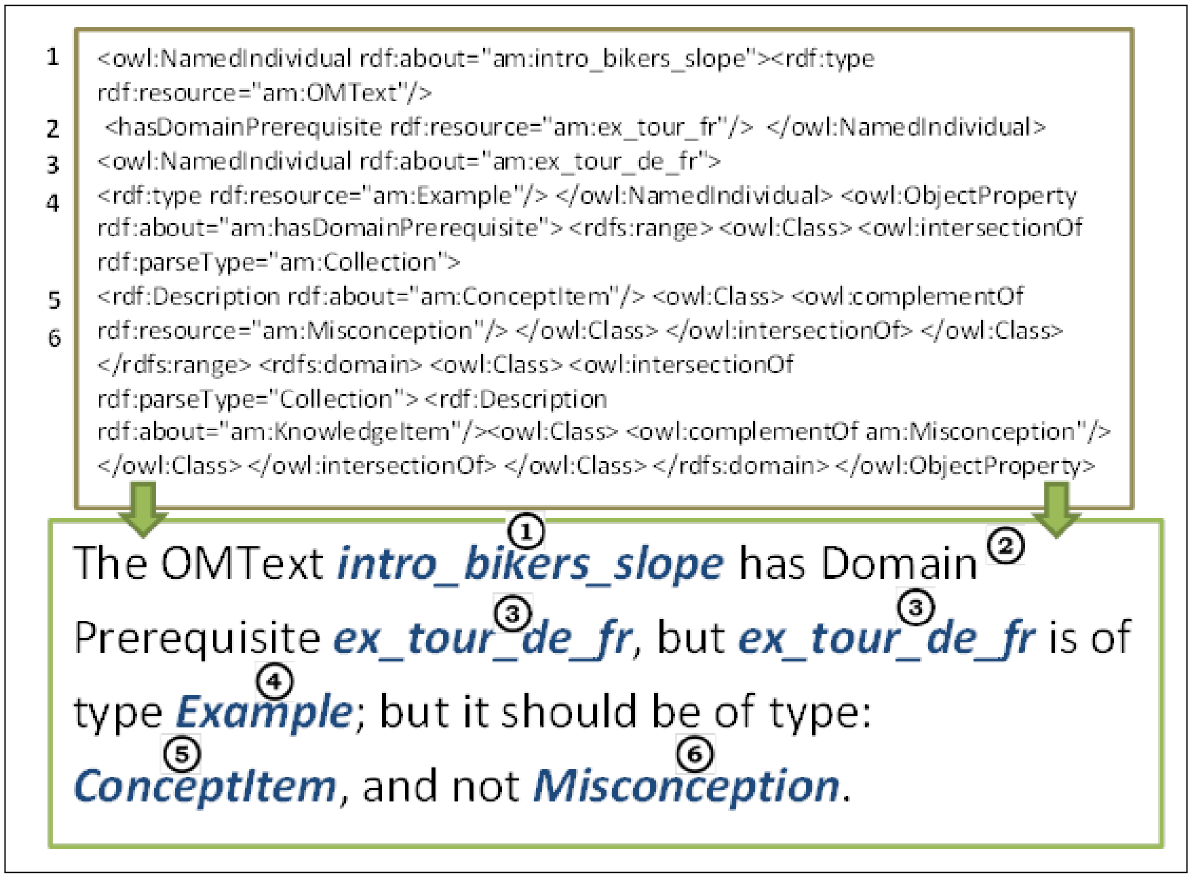
Figure 4.5: Isolation of axioms causing an ontology to be inconsistent

4.2.4 Generation of verbal explanations

The justifications for inconsistent ontologies, as seen before, are sets of OWL2 axioms, which can be difficult to understand for authors. This is why the final step of the approach is to generate verbal, human-friendly explanations for the metadata gaps in the collections.

Verbal explanations are generated by analyzing all axioms of a justification to extract the meaning of the gap they represent, and generate a verbal explanation for it. This analysis is also important because justifications can overlap each other, and represent the same gap. At the end, the resulting verbal explanations represent unique metadata gaps.

An example of this analysis is presented by Figure 4.6. The figure shows the verbal explanation for the isolated axioms showed in Figure 4.5. The *owl:NamedIndividual* (number 1) and *hasDomainPrerequisite* (number 2) axioms indicate that `'intro_bikers_slope'` uses the relation *hasDomainPrerequisite* to link with the object `'ex_tour_de_fr'` (number 3). Additionally, the *rdf:type* axiom indicates that `'ex_tour_de_fr'` is of type *Example* (number 4), but the *owl:property* axiom tell us that the relation *hasDomainPrerequisite* must link only to objects of type *ConceptItem* (number 5) and not *Misconception* (number 6); thus a verbal explanation for the metadata gap is generated to explain this logic chain.



```
1  <owl:NamedIndividual rdf:about="am:intro_bikers_slope"><rdf:type
   rdf:resource="am:OMText"/>
2   <hasDomainPrerequisite rdf:resource="am:ex_tour_fr"/>  </owl:NamedIndividual>
3  <owl:NamedIndividual rdf:about="am:ex_tour_de_fr">
4  <rdf:type rdf:resource="am:Example"/> </owl:NamedIndividual> <owl:ObjectProperty
   rdf:about="am:hasDomainPrerequisite"> <rdfs:range> <owl:Class> <owl:intersectionOf
   rdf:parseType="am:Collection">
5  <rdf:Description rdf:about="am:ConceptItem"/> <owl:Class> <owl:complementOf
6  rdf:resource="am:Misconception"/> </owl:Class> </owl:intersectionOf> </owl:Class>
   </rdfs:range> <rdfs:domain> <owl:Class> <owl:intersectionOf
   rdf:parseType="Collection"> <rdf:Description
   rdf:about="am:KnowledgeItem"/><owl:Class> <owl:complementOf am:Misconception"/>
   </owl:Class> </owl:intersectionOf> </owl:Class> </rdfs:domain> </owl:ObjectProperty>
```

The OMText ***intro_bikers_slope*** has Domain Prerequisite ***ex_tour_de_fr***, but ***ex_tour_de_fr*** is of type ***Example***; but it should be of type: ***ConceptItem***, and not ***Misconception***.

Figure 4.6: Example of a verbal explanation

The exemplified verbal generation process is done for each justification found in the learning object ontologies with inconsistencies in the collection, to generate clear verbal explanations for the authors.

Chapter 5

Implementation

The proposed approach was implemented as a Java program to check the metadata correctness in the learning object collections of ActiveMath. This chapter gives insight into the details of the two main functionalities of the tool: *transformation*, and *validation*. The former is responsible for transforming all the metadata in the content collections into OWL2 ontologies. The *Validation* phase checks the correctness of the metadata and generates verbal explanations when gaps are found.

5.1 General Description

The tool was developed as a Java console program. It receives three parameters:

- A string to indicate the main path in the file system containing the learning object collections to be processed.
- A boolean variable to denote if the learning object collections have to be converted to OWL2.
- A boolean variable which indicates if the metadata validation task should be executed.

Each of the two tasks is executed independently and in order: first transformation, and then validation. Both tasks use the given path to get the collections to be processed. Additionally, each task logs information about the progress and results.

The program uses the OWL Java API [9] to process OWL2 ontologies.

5.2 Transformation

This task is responsible for transforming all the OMDoc files from the learning object collections into OWL2 ontologies. It implements the Step 1 from the proposed approach (see Chapter 4, Section 4.2). The tool reads all the folders in the main path, and for each of them, converts all the OMDoc files inside the subfolder 'omdoc' to OWL2 ontologies. The new files are placed inside the subfolder 'owl'. Xalan-Java[1] is used as the XSLT processor for the transformations.

5.2.1 XSLT stylesheet

In terms of OMDoc, learning objects and their metadata are represented as XML elements and attributes. On the other hand, the metadata schema uses OWL2 classes to represent the learning objects and metadata elements, and OWL2 object properties to represent the metadata elements that relate learning objects to each other, and assign metadata values to the learning objects (see Figure 4.1). An XSLT stylesheet contains all the rules to transform the learning object metadata between the two formats.

The developed OMDoc-to-OWL2 XSLT stylesheet contains twenty four template rules, which are divided into four categories: *initial matching*, *learning objects*, *metadata*, *auxiliaries.*

Initial matching

Four template rules are used to match the *root* element of OMDoc, and other initial elements. With these rules, the XSLT processor knows that it is processing the right XML file and can continue with the rest of the document.

The rules in this category are:

xml First rule to match the beginning of an XML document.

[1]`http://xml.apache.org/xalan-j/`

omdoc Matchs the *omdoc* element, which indicates the start of an OMDoc document. In this rule, the creation of the OWL2 ontology starts with the definition of the element *rdf:RDF*. Additionally, the rule creates the import declaration for the metadata schema, which is necessary to perform the metadata validation. This rule is shown in Figure 5.1.

theory Learning objects are defined inside *theories*. This element indicates the start of a new *theory*. After this element, the rules for *learning objects* are applied.

omgroup Sometimes the elements in OMDoc are grouped using the *omgroup* element; this rule matches the elements under this one.

```
<xsl:template match="omd:omdoc">
       <rdf:RDF>
               <owl:Ontology>
                     <owl:imports>
                            <owl:Ontology
                               rdf:about="activemath.schema.owl"/>
                     </owl:imports>
               </owl:Ontology>
               <xsl:apply-templates />
       </rdf:RDF>
</xsl:template>
```

Figure 5.1: XSLT rule for the *omdoc* element

Learning objects

Ten rules convert the learning objects defined in OMDoc to the ones in the metadata schema. Several rules are necessary because there are different types of learning objects. Nine of this rules are one-to-one rules, which convert learning objects that are equivalent in OMDoc and the metadata schema. One rule is used to match a learning object in OMDoc into three different ones, according to the metadata schema.

one-to-one These rules match one element from OMDoc to another in OWL2. The rules process the follow learning objects: *symbol*, *definition*, *example*, *proof*, *exercise*, *axiom*, *omtext*, *ppmethod* (to *Method*), *cape* (to *Misconception*).

one-to-multiple This rule transform an *assertion* object, into three types of learning object: *Corrollary*, *Lemma*, and *Theorem*. The rule uses the `type` attribute to identify the correct learning object.

After each learning object has been matched to the corresponding rule, ‘metadata’ and ‘auxiliaries’ rules are applied to transform the metadata and descriptive information of the learning object into OWL2.

Figure 5.2 shows the XSLT rule to transform the *examples*.

```
<xsl:template match="omd:example">

        <!-- creat the instance-->
        <Example>
                <xsl:call-template name="idmatter"/>

                <!-- create the Object Properties (Domain)-->
                <xsl:call-template name="idrefmatter"/>

                <!-- create the Object Properties (Pedagogic), call to the
                   metadata template -->
                <xsl:apply-templates select="omd:metadata">
                        <xsl:with-param name="idElement" select="@id" />
                </xsl:apply-templates>
        </Example>

        <!-- create the Object Property (Pedagogic) "Learning Context" ,
           call to the template -->
        <xsl:apply-templates select="omd:metadata/omd:extradata" mode="lc">
                <xsl:with-param name="idElement" select="@id" />
        </xsl:apply-templates>

        <!-- call subclases -->
        <xsl:apply-templates select="omd:symbol"/>

</xsl:template>
```

Figure 5.2: XSLT rule for *examples*

Metadata

Five rules transform the metadata of the learning objects. OMDoc specifies the metadata fields as elements, and uses the `value` attribute to declare the value of such elements. Additionally, metadata elements that relate learning objects to each other are represented either using the *relation* element or the `for` attribute. Rules in this category manage the *relation* element, and rules in *auxiliaries* are in charge of the `for` attribute.

The rules for metadata are:

metadata Matches the *metadata* element. After this element, the 'extradata' rule is applied.

extradata This rule matches the *extradata* element, which contains the pedagogic and semantic metadata. This rule assigns in OWL2 to the corresponding learning object the next metadata elements: *abstractness*, *field*, *learningcontext*, *difficulty*, *exercise-type*, *competence-level*, *competence*, and *representation*. Also, the rule matches the metadata elements for learning object relations under the *relation* element. Examples of such relations are *conclusion for*, *is domain prerequisite for* and *is motivation for*. All this semantic relations are defined as object properties in the metadata schema. Both metadata elements for semantic relations and metadata values are assigned to the learning objects using the proper object property.

metadataLearningContext Special rule to create correctly the *LearningContext* of each learning object.

addOPPedagogic Used by the 'extradata' rule to create the pedagogic metadata.

addXref The 'extradata' rule uses it to process the attributes in the *relation* element.

A part of the 'extradata' rule matching the *relation* element is shown in Figure 5.3.

Auxiliaries

Finally, these five rules are used by other rules to read the attributes of the elements in OMDoc.

In this category the rules are:

idmatter To extract the id of the learning object from the `id` attribute.

idgmatter Similar to 'idmatter' rule, but it also matches the `generated-by` attribute.

```
<xsl:for-each select="am:relation">
 <xsl:variable name="typeR" select="@type"/>
 <xsl:for-each select="omd:ref">
       <xsl:choose>
              <xsl:when test="$typeR = 'conclusion_for' ">
                     <isConclusionFor>
                            <xsl:call-template name="addXref"/>
                     </isConclusionFor>
              </xsl:when>
              <xsl:when test="$typeR = 'is_domain_prerequisite_for' ">
                     <isDomainPrerequisiteFor>
                            <xsl:call-template name="addXref"/>
                     </isDomainPrerequisiteFor>
              </xsl:when>
              (...)
</xsl:for-each>
```

Figure 5.3: Part of the XSLT rule matching the *relation* element in OMDoc

idrefmatter, forImplied These rules create relations among the learning objects, such as *isMotivationFor*, *icConlusionFor*, and *isProofFor*, by using the value of the `for` attribute.

addOPDomain This rule is used by the 'forImplied' and 'idrefmatter' rules to read the `for` attribute.

A part of the 'forImplied' rule matching the `for` attribute for the *exercise* element is shown in Figure 5.4.

5.3 Validation

This functionality of the tool corresponds to the Steps 2, 3 and 4 of the proposed approach (see Chapter 4, Section 4.2). The main purpose of the task is to compute all the gaps in the content and keep statistics on the number and types of gaps for each of the learning collections. The tool processes each learning object ontology in the 'owl' subfolders from

```
<xsl:template name="forImplied">
       <xsl:if test="@for">
              <xsl:choose>
                     <xsl:when test=". = //omd:exercise">
                            <isExerciseFor>
                                   <xsl:call-template
                                      name="addOPDomain"/>
                            </isExerciseFor>
                     </xsl:when>
                     (...)
              </xsl:choose>
       </xsl:if>
</xsl:template>
```

Figure 5.4: Part of the XSLT rule matching the `for` attribute in OMDoc

the collections in the main path.

Each learning object ontology is processed in a sequence of steps, which are explained below.

5.3.1 Pre-processing

Sometimes metadata relations relate to learning objects, which are defined in different learning object ontologies. Because of that, a pre-processing is necessary to check if those learning objects exist, and then to include them in the current ontology. If there are learning objects that are not part of the current learning object ontology, the reasoner could not find all the possible gaps. External learning objects are referenced first using the id of the theory where they are defined, and then the id of the learning object.

The program scans the complete learning object ontology, and each time a reference to an external learning object appears, it finds the ontology where the external theory is defined. If the theory could not be found, it generates a new gap of type *External theory does not exist* (see subsection 5.3.4). If the theory exists, the program opens the ontology and verify that the learning object exists. If it does exist, the type of the learning object is extracted and included in the current ontology. In the other case, if the learning object does not exist, the program creates a new gap of type *External learning object does not exist* (see subsection 5.3.4). After all external references have been checked, inconsistencies are detected.

5.3.2 Ontology inconsistencies

HermiT[2] is used as the semantic reasoner to detect ontology inconsistencies. If the reasoner detects conflicts in the learning object ontology, the tool moves to the next step to detect the exact errors. In the other case, the ontology is marked as correct. The structure of the learning object ontologies follows the structure of the OMDoc documents in the ActiveMath learning collections, which usually defines several learning objects in each file. This means that the consistency check indicates if there are or not gaps for any of the learning objects in the learning object ontology. If it is inconsistent, each learning object is analyze individually in the next step to detect the precise gaps.

5.3.3 Isolation of axioms

When a learning object ontology has inconsistencies, it is necessary to detect which metadata is causing the errors. This is done using the OWL API and the reasoner to detect the gaps in the ontology, but this has practical implications because of the running time to detect such gaps, which makes impossible to get all the gaps (in a reasonable amount of time) for a reduced number of learning objects.

The program reads the ontology, and for each learning object, it creates a temporal ontology containing the current learning object and its metadata. Then, it tries to compute all the gaps for the learning object. If the algorithm is still running after twenty minutes, the learning object is marked as skipped. This problem can happen because there are learning objects with a lot of metadata and gaps. Although the technique is not perfect,

[2]`http://www.hermit-reasoner.com/`

it has proven to be successful. The program found all the gaps for 100% of the learning objects in five learning collections in ActiveMath, and around 99% in two other collections (see Chapter 6).

At this stage of the process, the found gaps are represented as sets of OWL2 axioms causing logical errors in the learning object ontology.

5.3.4 Verbal explanation

As explained before, the axioms representing a metadata gap are difficult to understand for authors. To solve this problem, the tool takes the set of axioms of a gap - generated in the step before - and converts them into a human-readable verbal explanation.

Axioms in OWL2 have different types, each representing particular declarations or properties in the ontologies. For example, an *ObjectPropertyAssertion* axiom indicates a relation and the learning objects that form part of it. The *DisjointObjectProperties* axiom indicates two relations that cannot be applied to the same learning objects, because they are disjoint. The program analyses the axioms according to their types and the information they represent to produce a verbal explanation. Based on an analysis of the learning collections in ActiveMath, eight types of gaps were identified:

1. **External learning object does not exist (ELO)**: This gap occurs when a learning object metadata element references another learning object from an external ontology, but the external learning object does not exist.

2. **External theory does not exist (ETHEORY)**: This gap happens when a learning object metadata element references another learning object from an external ontology, but the external theory does not exist.

3. **Undefined learning object or metadata value (UNDEFINED)**: This error takes place when a a learning object metadata element references another learning object that does not exist, or when a metadata element has an undefined value.

4. **Domain of a metadata element is wrong (DOMAIN)**: This gap occurs when a learning object uses a metadata element, but according to the domain restrictions in the metadata schema, the learning object cannot utilize such metadata element.

5. **Range of a metadata element is wrong (RANGE)**: This error takes place when a learning object uses a metadata element, but according to the range restrictions in the metadata schema, the element value is an incorrect type of learning object or metadata value.

6. **Metadata element is functional (FUNCTIONAL)**: This problem is caused when a learning object applies the same metadata element more that once, but according to the metadata schema, it can be applied only one time.

7. **Learning object or metadata value type definition (TYPE)**: This gap happens when a learning object or metadata value is defined two or more times. Each learning object or metadata value is unique and can not have multiple definitions.

8. **Metadata elements are disjoint (DISJOINTOR)**: This error occurs when two learning objects are related by two metadata elements, but those elements are disjoint, i.e. they cannot relate the same two learning objects.

When the program detects that the axioms correspond to a specific type of gap, it generates a verbal explanation for it, based on predefined templates. Table 5.1 shows templates for each type of gaps along with examples. In the *Template for error message* column, LO1 refers to the learning object that is using a metadata relation, and the learning object that the relation links to is represented as LO2.

Table 5.1: Verbal explanations for gaps

Gap	**Template of error message**	**Example**
External learning object does not exist	[ELO] The LO1:type [LO1:id] references the external learning object [theory/LO2:id], but this learning object does not exist in the specified theory.	[ELO] The Proof [prf_tangent] references the external learning object [lines/form_point_slope], but this learning object does not exist in the specified theory.
External theory does not exist	[ETHEORY] The LO1:type [LO1:id] references the external learning object [theory/LO2:id], but the specified theory does not exist.	[ETHEORY] The Example [ex0_applet_diff_f] references the external learning object [calculus1/diff], but the specified theory does not exist.
Undefined learning object or metadata value	[UNDEFINED] The learning object or metadata value [LO2:id] is not defined in the content, but it is referenced or used.	[UNDEFINED] The learning object or metadata value [tour_f_example] is not defined in the content, but it is referenced or used.

Continued on next page

Table 5.1 – *Continued from previous page*

Gap	**Template of error message**	**Example**
Domain of a metadata element is wrong	[DOMAIN] The LO1:type [LO1:id] metadata-relation:name [LO2:id or metadata-value:value], but [LO1:id] cannot use this relation. The relation can be only used by learning objects of type: metadata-relation:domain	[DOMAIN] The Definition [defX_slope] has Field [all], but [defX_slope] cannot use this relation. The relation can be only used by learning objects of type: Example, or SateliteItem.
Range of a metadata element is wrong	[RANGE] The LO1:type [LO1:id] metadata-relation:name [LO2:id or metadata-value:value], but [LO2:id or metadata-value:value] is of type: LO2:type or metadata-value:type; but it should be of type: metadata-ralationship:range.	[RANGE] The OMText [intro_act_slope] is Introduction For [diffquot/intro_hiking], but [diffquot/intro_hiking] is of type: OMText; but it should be of type: ConceptItem.

Continued on next page

Table 5.1 – *Continued from previous page*

Gap	**Template of error message**	**Example**
Metadata element is functional	[FUNCTIONAL] The LO1:type [LO1:id] uses the relation metadata-relation:name, but this relation is functional, and therefore it can be only applied one time for each learning object.	[FUNCTIONAL] The Example [ex_diff_parab] uses the relation has Abstractness, but this relation is functional, and therefore it can be only applied one time for each learning object.
Learning object or metadata value type definition	[TYPE] The learning object or metadata value [LO1:id or metadata-value:value] is of type: LO1:type or metadata-value:type, but it should have only one of those types.	[TYPE] The learning object or metadata value [drag_and_drop] is of type: ExerciseType and InteractivityType, but it should have only one of those types.

Continued on next page

Table 5.1 – *Continued from previous page*

Gap	**Template of error message**	**Example**
Metadata elements are disjoint	[DISJOINTOR] The LO1:type [LO1:id] uses the relations [metadata-relation1:name, metadata-relation2:name] with the LO2:type [LO2:id], but those relations or the inverse ones are disjoint, i.e. they cannot both relate the same learning objects.	[DISJOINTOR] The Example [ex_quad_diff] uses the relations [has Domain Prerequisite, is Example For] with the Symbol [deriv_symbols/diff], but those relations or the inverse ones are disjoint, i.e. they cannot both relate the same learning objects.

Chapter 6

Evaluation and Results

The effectiveness of the proposed approach means not only that it can find a large number of gaps, but also, that the approach can detect the gaps that the expert content authors can find manually. This chapter presents the results obtained from the validation of metadata from seven learning collections in ActiveMath. Additionally, it describes a comparison procedure that was designed to test the effectiveness of the tool against manually debugged versions of the learning object collections.

6.1 Gaps in ActiveMath

The developed tool was used to find gaps in seven learning collections in ActiveMath. These collections were developed in different universities and institutes around Europe, for example, the *University of Kassel*[1] in Germany and the *Open Universiteit*[2] in the Netherlands; and together represent more than 12 000 learning objects.

Table 6.1 presents the statistics of the number of gaps by collection. The third column indicates how many learning objects in that collection the program skipped, because it took too much time to compute their gaps . The fourth one displays the percentage of learning objects with all the gaps computed. The last column indicates the number of unique gaps in each collection. The number of unique gaps represents the amount of gaps that denote the same error in a conceptual way. For example, if an author applies the 'exerciseType' metadata element to five different *examples*, this produces five different gaps, but they are

[1]`http://www.uni-kassel.de`
[2]`http://www.ou.nl`

counted as just one unique gap, because all of them represent the same conceptual gap.

Table 6.1: Number of gaps by collection

Collection	**# of LO's**	**# of skipped LO's**	**% of analyzed LO's**	**# of gaps**	**# of unique gaps**
BasWis	1342	1	99,93%	734	42
kasselContent	2736	0	100,00%	798	44
kwadratvg	32	0	100,00%	10	3
LeAM_calculus	6176	0	100,00%	987	193
ounlContent	188	0	100,00%	197	13
schulmathematik	2321	8	99,66%	1532	21
tutEx	138	0	100,00%	69	1

For each collection, the number of gaps by type was also computed. These numbers are shown in Table 6.2.

Table 6.2: Detailed number of gaps by type: 1. ELO, 2. ETHOERY, 3. UNDEFINED, 4. DOMAIN, 5. RANGE, 6. FUNCTIONAL, 7. TYPE, 8. DISJOINTR

Collection	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
BasWis	0	0	47	1	683	0	2	1
kasselContent	0	0	476	100	194	7	0	21
kwadratvg	0	0	0	0	10	0	0	0
LeAM_calculus	178	176	64	82	356	32	0	99
ounlContent	0	4	187	0	1	0	0	5
schulmathematik	0	1	57	1001	424	0	0	49
tutEx	0	0	0	69	0	0	0	0

6.2 Effectiveness

Evaluation was performed in order to determine if the developed tool can detect the same gaps as a content author would do. A comparison functionality was added to the tool, which compares two different versions of the same collection of learning objects, where some quality improvement tasks were made in the first version over time, resulting in the second version. The changes in the collection were made by expert content authors, in this way, it is possible to compare the gaps in both collections, and see if the tool can detect gaps in the first version of the collection that were fixed by authors in the later one. ActiveMath uses a SVN system to keep the versioning of the learning collections, which makes possible and easy to retrieve the different versions to be compared.

6.2.1 Procedure

Figure 6.1 shows the process to compare two versions of the same learning object collection. The first step is to select a list of pairs of versions of the learning collections to compare. Two versions of the 'kasselContent' learning collection were manually selected. This collection is ideal for this evaluation because, it has a significant number of learning objects that can be compared (1299).

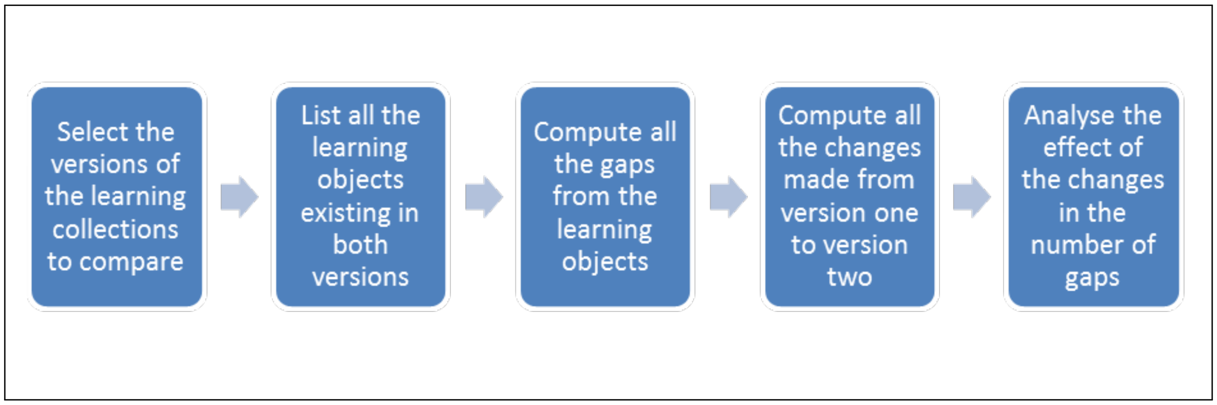

Select the versions of the learning collections to compare
List all the learning objects existing in both versions
Compute all the gaps from the learning objects
Compute all the changes made from version one to version two
Analyse the effect of the changes in the number of gaps

Figure 6.1: Comparison process between versions of a learning collection

After the versions have been selected, it is necessary to create a list with the learning objects that will be compared. Because different version of the same collection differ not only in the metadata but also in the number of learning objects, only learning objects that exist in both versions are selected. This is because using different learning objects that exist only in one of the versions of the collection will not produce a reliable point of comparison. The next step is to compute all the gaps from the selected learning objects in both versions of the learning collections.

The fourth step corresponds to the computation of all the changes made in the learning objects, in order to get from Version 1 to Version 2. Each change is classified in one of four categories:

1. **Update**: when a metadata value is updated from Version 1 to the value in Version 2.

2. **deletion**: when metadata is deleted from Version 1; therefore, the value is not presented in Version 2.

3. **addition 1 (old LO's)**: when new metadata is introduced in Version 2 of the learning object. In this case, the new metadata is associated to learning objects that were first defined in the Version 1, and that still exist in Version 2 of the collection.

4. **addition 2 (new LO's)**: when new metadata is introduced in Version 2 of the learning object, but in this case, the new metadata connects to new learning objects, which are first defined in the Version 2 of the collection.

All the metadata that is the same in both versions of the collection is classified as **remaining**. In order to avoid the problem of classifying an update, first as one deletion and then as one addition, the changes are computed in a specific order: first all the unchanged metadata, then the updates, third the deletions, and finally the additions.

The last step is to apply all the changes from the previous step, one by one to the first version in order to reconstruct Version 2, and record the effect that those changes have in the total number of gaps in the learning objects. All the changes are applied in a specific order, starting with updates, continuing with deletions, and ending with additions. After a change is applied, the number of gaps is recomputed, and according to the effect of the change in the number of gaps, it is classified as:

1. **decrease**: if the change produces one or more gaps to be fixed.
2. **increase**: if the change produces one or more new gaps.
3. **no effect**: if the change has not effect in the number of gaps.

6.2.2 Results

The comparison process was applied to the *kasselContent* learning collection. Table 6.3 presents the number of changes in the metadata, and the effect of those changes in the total number of gaps in the collection. The same data is represented graphically as a bar char in Figure 6.2.

Table 6.3: Effect in the number of gaps by type of changes

Effect in number of gaps	**Type of change**				
	deletions	*updates*	*additions 1 (old LO's)*	*additions 2 (new LO's)*	*total*
decrease	595	936	0	0	1531
increase	0	3	21	344	368
no effect	260	283	1600	2452	4595
Total	855	1222	1621	2796	6494

Version 1 of the collection has 1951 gaps, and version 2 has 789 gaps. This means

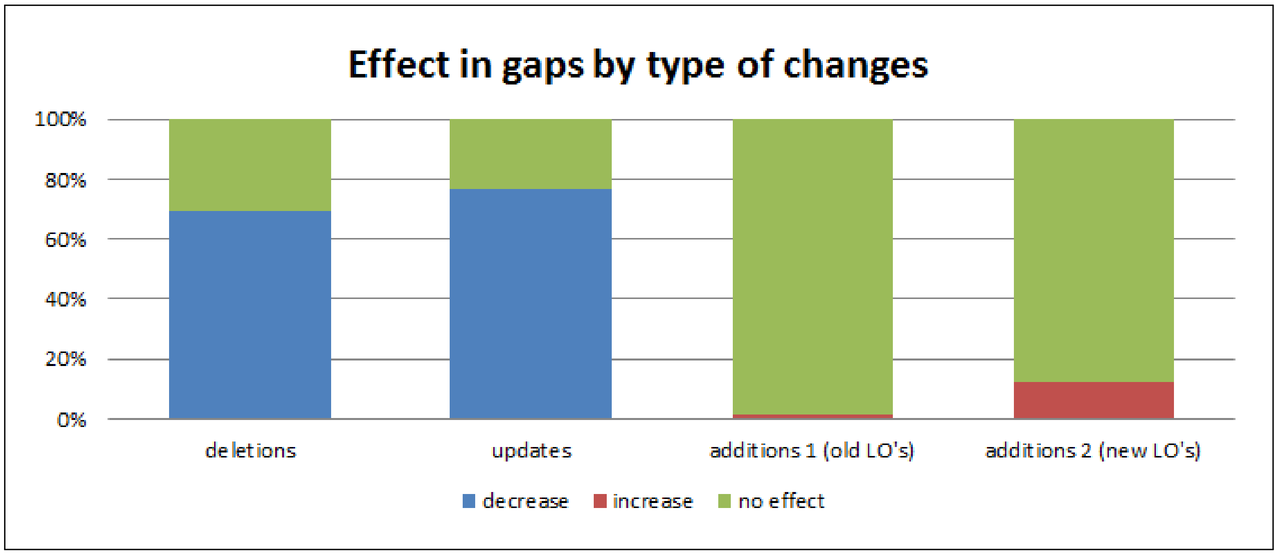


Figure 6.2: Chart of the effect in the number of gaps by type of changes

that 1531 gaps from Version 1 were fixed, 420 gaps were remaining, and 369 gaps were introduced in the second version of the learning collection. From the total number of new gaps, 25 are caused by changes of type addition 1 (old LO's) and 344 by additions 2 (new LO's). This information is presented in the pie charts in Figure 6.3 and Figure 6.4.

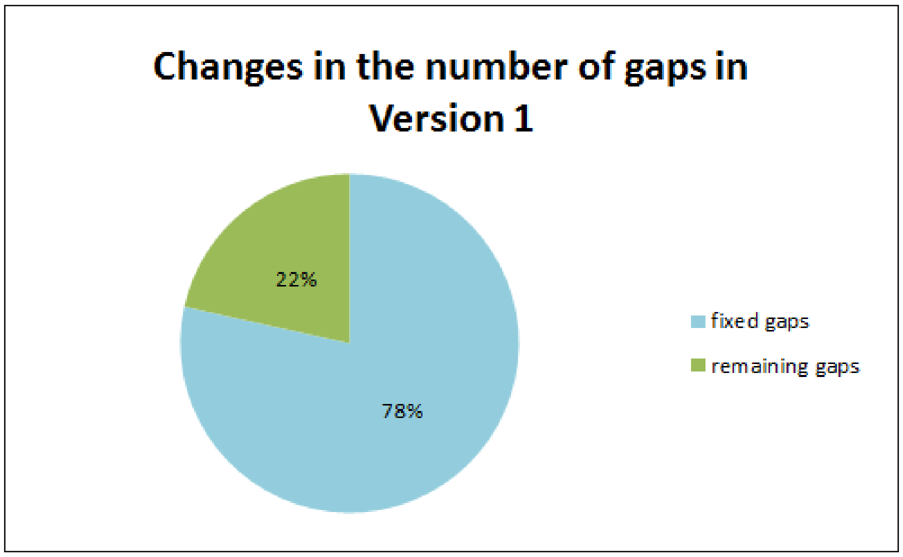


Figure 6.3: Changes in the number of gaps in the Version 1 of the collection

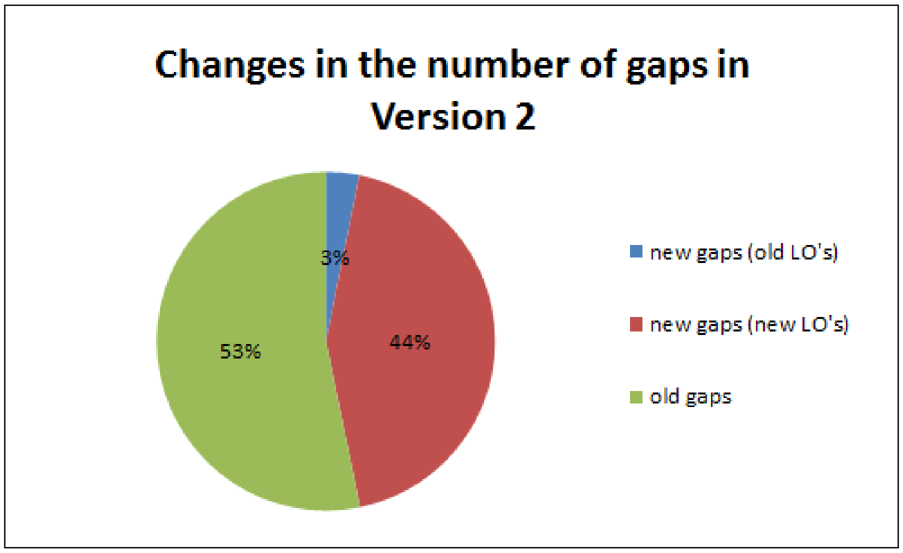


Figure 6.4: Changes in the number of gaps in the Version 2 of the collection

6.3 Discussion

The results of the evaluation show that Version 2 of the collection has significantly fewer number of gaps that Version 1 (789 against 1951). This indicates that changes were not only made to add metadata, but also to fix existing gaps in the earlier version of the content, and therefore, increase the quality of the collection. From the changes in the metadata (see Table 6.3), it is concluded that the largest number of updates (936/1222) and deletions (595/855) were introduced in Version 2 in order to fix gaps, and that the greatest number of new gaps (344/369) in the content were caused by new added metadata that relates to new learning objects in the latest version of the collection. In terms of all the changes that did not modify the number of gaps (4595/6494), they were done to increase the amount and quality of metadata in the learning objects or to fix semantic gaps that the tool could not detect.

From the data in Figure 6.2, three conclusions can be drawn: *a*) the tool was capable to detect the same gaps that the authors did (the blue area); *b*) the tool detected additional gaps which were not found by the authors (the red area); and that *c*) the tool detected newly added metadata which is free of gaps or with gaps that neither the tool nor the authors detected (the green area). This metadata could also have been introduced to solve gaps in the Version 1 that the tool did not detect. Based on this analysis, it is not possible to confirm that the tool detects 100% all the existing gaps. In order to get this information a manual evaluation by a group of expert authors is necesarry. This analysis may show that some changes in the collection were made to fix invisible gaps to the tool. This task is part of the future work of this thesis (see Chapter 7).

Finally, it is possible to conclude that the tool effectively detects semantic gaps in the content, and that it can be used by content authors to improve the quality of the metadata in the learning objects.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis presents a robust mechanism to detect automatically inconsistencies in the learning object metadata. The metadata is represented in OWL2 format and then validated against a metadata schema using a semantic reasoner. This validation shows formal conflicts presented in the content. These conflicts represent gaps in the metadata, which are presented to the authors in the form of human-readable messages.

Many metadata standards use XML representation, and even when this language offers little support for validation, the proposed approach presents a method to perform metadata validation despite this restriction. Additionally, any metadata standard can be adapted to the proposed approach with little effort. Another feature of the approach is that cheeking if a learning object has gaps takes only a few seconds. This allows to develop an interactive framework, where the authors can check if a learning object has gaps, receive explanations for the detected gaps, and then fix the gaps. This interaction will continue until the learning objects are free of gaps.

The proposed approach has been implemented as a tool to validate the learning content of ActiveMath, an intelligent tutoring system for mathematics. The tool generates eight different types of gaps, which allows the content authors to identify the gaps easily. An evaluation has been performed to determine the effectiveness of the tool. Results indicate that the tool can effectively detect gaps in learning object collections. Furthermore, the evaluation showed that the tool detected gaps that the content authors did not, but it was not possible to conclude that the tool detected all the gaps that exist in the content. A manual evaluation needs to be carried out to obtain more significant results.

7.2 Future Work

The future work is planned in the following directions:

1. *Integrate the developed tool with the Authoring Tool in ActiveMath.* The developed gap detection tool needs to be integrated as a component to the new authoring tool in ActiveMath. This component will allow to check if any learning object has gaps, and if it does, generate through an interactive process the descriptions of the existing gaps. In this process, the component first generates at most ten gaps every time the authors ask for gaps, then such gaps can be fixed, and if the learning object still has errors, the tool generates new explanations. This process is needed because generating all the gaps for a learning object could take too much time, or overwhelm the authors with a large number of gaps to be fixed.

2. *Refine the metadata requirements in the Metadata Schema.* The developed tool only can find gaps according to the metadata requirements and restrictions that are defined in the metadata schema. Although the ActiveMath metadata schema has a lot of requirements, new restrictions must be added in order to make the schema most robust. For example, the *cardinality* restriction to indicate that certain metadata value must exist for a learning object is not defined.

3. *Manually analyze the learning content.* A manual evaluation is necessary to measure the effectiveness of the developed tool. Expert content authors can take a learning object collection and find all the existing gaps, and then compare the results with the ones obtained using the gap detection tool. This evaluation could determine if the tool can detect all the gaps that exist in the learning object collections.

References

[1] Franz Baader and Boontawee Suntisrivaraporn. Debugging snomed ct using axiom pinpointing in the description logic el+. In *KR-MED*, 2008.

[2] Beatriz Barros, Felisa Verdejo, Timothy Read, and Riichiro Mizoguchi. Applications of a collaborative learning ontology. In *Proceedings of the Second Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence*, MICAI '02, pages 301–310, London, UK, UK, 2002. Springer-Verlag.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[4] Ronald Denaux, Vania Dimitrova, Anthony G. Cohn, Catherine Dolbear, and Glen Hart. Rabbit to owl: ontology authoring with a cnl-based tool. In *Proceedings of the 2009 conference on Controlled natural language*, CNL'09, pages 246–264, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] Gary Geisler, Sarah Giersch, David McArthur, and Marty McClelland. Creating virtual collections in digital libraries: benefits and implementation issues. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, JCDL '02, pages 210–218, New York, NY, USA, 2002. ACM.

[6] W3C OWL Working Group. Owl 2 web ontology language document overview. `http://www.w3.org/TR/owl2-overview/`.

[7] W3C OWL Working Group. Owl 2 web ontology language primer. `http://www.w3.org/TR/owl2-primer/`.

[8] Marek Hatala and Griff Richards. Value-added metatagging: Ontology and rule based methods for smarter metadata. In *In Rules and Rule Markup Languages for the Semantic Web (RuleML2003)*, pages 65–80, 2003.

[9] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semant. web*, 2(1):11–21, January 2011.

[10] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Explaining inconsistencies in owl ontologies. In Lluis Godo and Andrea Pugliese, editors, *SUM*, volume 5785 of *Lecture Notes in Computer Science*, pages 124–137. Springer, 2009.

[11] Mitsuru Ikeda, Kazuhisa Seta, and Riichiro Mizoguchi. Task ontology makes it easier to use authoring tools. In *In Proc. of the 15th IJCAI*, pages 342–347. Morgan Kaufmann Publishers, Inc, 1997.

[12] Dominik Jednoralski, Erica Melis, Sergey Sosnovsky, and Carsten Ullrich. Gap Detection in Web-Based Adaptive Educational Systems. In Xiangfeng Luo, Marc Spaniol, Lizhe Wang, Qing Li, Wolfgang Nejdl, and Wu Zhang, editors, *Advances in Web-Based Learning ICWL 2010*, volume 6483 of *Lecture Notes in Computer Science*, pages 111–120, Berlin, Heidelberg, 2010. Springer Berlin / Heidelberg.

[13] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of owl dl entailments. In *ISWC/ASWC*, pages 267–280, 2007.

[14] Michael Kohlhase. Omdoc: Towards an internet standard for the administration, distribution and teaching of mathematical knowledge. In *IN PROCEEDINGS AISC'2000*, pages 32–52. Springer Verlag, 2000.

[15] Michael Kohlhase. *OMDoc – an Open Markup Format for Mathematical Documents: (version 1.2)*. Lecture Notes in Artificial Intelligence. Springer, 2006.

[16] T. Leacock, H. Farhangi, A. Mansell, and K. Belfer. Infinite possibilities, finite resources: The techbc course development process. In *Proceedings of the 4th Conf. on Computers and Advanced Technology in Education (CATE 2001)*, 2001.

[17] Catherine C. Marshall. Making metadata: a study of metadata creation for a mixed physical-digital collection. In *Proceedings of the third ACM conference on Digital libraries*, DL '98, pages 162–171, New York, NY, USA, 1998. ACM.

[18] Erica Melis, Giorgi Goguadze, Paul Libbrecht, and Carsten Ullrich. Activemath – a learning platform with semantic web features. In *Ontologies and Semantic Web for e-Learning*. IOS Press, 2009. in print.

[19] Riichiro Mizoguchi, Yusuke Hayashi, and Jacqueline Bourdeau. Inside Theory-Aware and Standards-Compliant Authoring System. In Harrer Andreas Capuano Nicola, Dicheva Darina and Mizoguchi Riichiro, editors, *SW-EL'07 @ AIED'07 - Fifth International Workshop on Ontologies and Semantic Web for E-Learning*, pages 1–18, Marina del Rey, CA, USA, 2007.

[20] Olivier Motelet and Nelson Baloian. Hybrid system for generating learning object metadata. In *Proceedings of the Sixth IEEE International Conference on Advanced Learning Technologies*, ICALT '06, pages 563–567, Washington, DC, USA, 2006. IEEE Computer Society.

[21] Olivier Motelet, Nelson Baloian, and José A. Pino. Taking advantage of metadata semantics: the case of learning-object-based lesson graphs. *Knowl. Inf. Syst.*, 20(3):323–348, August 2009.

[22] Olivier Motelet, Benjamin Piwowarski, Georges Dupret, Jose A. Pino, and Nelson Baloian. Enhancing educational-material retrieval using authored-lesson metadata. In *Proceedings of the 14th international conference on String processing and information retrieval*, SPIRE'07, pages 254–263, Berlin, Heidelberg, 2007. Springer-Verlag.

[23] N. Press. *Understanding Metadata.* National Information Standards Organization Press, 2004.

[24] Vilho Raatikka and Eero Hyvönen. Ontology-Based Semantic Metadata Validation. *Towards the semantic web and web services Proceedings of XML Finland 2002 Conference*, (2002-03):28–40, 2002.

[25] Marko A. Rodriguez, Johan Bollen, and Herbert Van De Sompel. Automatic metadata generation using associative networks. *ACM Trans. Inf. Syst.*, 27(2):7:1–7:20, March 2009.

[26] Paramjeet Singh Saini, Marco Ronchetti, and Diego Sona. Automatic generation of metadata for learning objects. In *Proceedings of the Sixth IEEE International Conference on Advanced Learning Technologies*, ICALT '06, pages 275–279, Washington, DC, USA, 2006. IEEE Computer Society.

[27] Sergey Sosnovsky. Math-bridge: Metadata cookbook. `http://project.math-bridge.org/downloads/outcomes/deliverables/D1.1A-cookbook.final.pdf`.

[28] Sergey Sosnovsky. *Ontology-based open-Corpus Personalization for e-Learning.* dissertation, University of Pittsburgh, School of Information Sciences, 2011.

[29] P. Suraweera, A. Mitrovic, B. Martin, J. Holland, N. Milik, K. Zakharov, and N. McGUIGAN. Using ontologies to author constraint-based intelligent tutoring systems. *The Future of Learning*, page 77, 2009.