Constraint-based Validation of e-Learning Courseware

Mark Melia

Bachelor of Science in Software Systems

A Dissertation submitted in fulfilment of the

requirements for the award of

Doctor of Philosophy (Ph.D.)

to the

Dublin City University

Faculty of Engineering and Computing, School of Computing

Supervisor: Dr. Claus Pahl

September, 2009

Declaration

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

Signed:

[illegible]

Student ID: 50057436

Date: September 18, 2009

Contents

Abstract

Today e-learning courses, known as courseware, are expected to provide a highly engaging learning experience that adapts to a learner's needs and is interoperable with the learner's learning environment. This makes finding problems in the courseware increasingly difficult, tedious and expensive. A course creator may attempt to validate courseware manually by simulating a variety of possible learner paths through the courseware, but this is very laborious and time-consuming, and in most instances does not consider all possible variances in the courseware.

Courseware validation automatically checks that courseware conforms to a set of requirements and limits the risk of problems at delivery time. Traditional approaches to validation simulate learner instances going through a given courseware. This can lead to performance problems in highly adaptive courseware due to the complexity of simulation. These approaches are also inflexible as validation criteria are defined in software programming logic.

Courseware specifications define courseware in terms of its components. The componentisation of courseware into a collection of annotated Learning Objects (LOs) presents an opportunity to validate courseware based on its compositional structure. In this thesis, we present a novel approach to courseware validation based on Model Driven Engineering (MDE), using the component structure of courseware. We define a set of Domain Specific Language (DSL), known as the Courseware Authoring Validation Information Architecture (CAVIAr), and show how it can be used to capture courseware construction concerns. We then demonstrate how, by defining model constraints on a coursewares compositional structure, it allows for a flexible approach to courseware validation. We also investigate how CAVIAr-based tools can be integrated with the state of the art in e-learning. Our approach has been validated using a software implementation that allows course creators to validate courseware using CAVIAr.

Acknowledgements

First of all I would like to express my huge gratitude to my supervisor, Dr. Claus Pahl, for his patience, guidance and support. Thank you Claus for everything, you have been an exemplary supervisor and I have truly enjoyed working with you.

In the course of my research, I have been amazed at the generosity of senior researchers and academics with their time and knowledge. Many researchers in the Technology Enhanced Learning field provided me with invaluable feedback and help throughout my research. One person who I must mention is Dr. Dragan Gašević. All through my Ph.D. Dragan has never been too busy to help me out, sharing his immense wisdom with me.

I would like to thank my parents, Yvonne and Brendan, for their support and guidance throughout my (many) years of schooling. Nothing I have achieved could have happened without them and their belief in me. Mam and Dad thank you for always having time, from helping me with my spellings in Scoil San Carlo to those nights looking at project presentations. You have been a great example for me to live up to. Also, my grandmother and grandfather, Pat and June, who have encouraged me throughout not only my college years but also my school years, and by providing me with my “second” workplace. Thanks nanny and granda, I promise you can have the back bedroom back now. I am endepted to you all, I hope you know how much I appreciate all you have done for me.

Thanks to all my CA friends and colleagues past and present, who at this point are too numerous to name, but there are a couple who I have to mention. Firstly, I want to thank all the members of the OntAWare and MIKAEL projects, Edmond, Declan, Wong and Mo. In particular I would like to thank Wong and Mo, who helped me bring the MIKAEL tool to reality. Thanks guys for all the effort you put in, much appriciated. I would also like to thank Veronica and Claire, for watching all my presentations and providing me with insightful feedback.

A special mention has to go to Dr. Ronan Barrett. Ronan is an incredibly talented and devoted software engineer and researcher, and has always been there for encouragement, insightful conversation, and banter. Ronan thanks for everything, your input, advice, feedback and friendship.

Finally, I would like to thank my girlfriend Clare, who has been a rock of support. Clare, you never had any doubt in your mind that I would be able to do this, even when I tried to convince you otherwise. Thank you for your love, patience, encouragement and faith.

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Context

Courseware defines a course in terms of its learning content, by defining what learning content to deliver to a learner, when it should be delivered and how. Courseware is generally delivered and managed using some type of Virtual Learning Environment (VLE) or Learning Management System (LMS). Courseware authoring, also known as courseware construction, is a rapidly evolving research area that is concerned with the tools and methodologies, a course creator uses, to define and create courseware.

Understanding how people learn and using that knowledge to create better courses has always been a challenge for course creators. The literature provides the course creator with general course construction methodologies, embedded in pedagogical principles [Gagné et al., 2005, Reigeluth, 1999a, Reigeluth, 1983a, Briggs et al., 1991]. Course creators may also apply their own pedagogy in courseware construction. For example sequencing courseware topics in a particular way or ensuring that learning resources do not take an unreasonable amount of time to complete. Although course creators aim to apply pedagogical principles to the courseware they construct, when developing large courseware this can be difficult, especially when there are seemingly more pressing issues, such as standards compliance and delivery deadlines. When pedagogical principles are not adequately applied in courseware it can can lead learner confusion, motivational problems, learner isolation and ultimately to the rejection of the courseware [Samples, 2002]. Ensuring the application of pedagogical principles is therefore of paramount importance. To this effect the literature highlights the importance of post-construction course validation or auditing as an essential part of a holistic courseware construction methodology [Rosmalen et al., 2006, Persico, 1996]. Courseware validation ensures there are no problems in the courseware and pedagogical principles, as defined in the literature and by the course creator, have not been neglected and have been applied correctly.

Recently there has been a move towards adaptive or personalised courseware. This is courseware that moves away from the "one size fits all" paradigm of traditional courseware and instead looks to adapt to the individual needs of learners in terms of their learning goals, knowledge of a subject and their learning style, amongst other things [DeBra and Calvi, 1998, Eklund and Brusilovsky, 1999]. The construction of adaptive or personalised courseware is a "very complex, time consuming and expensive task" [Dagger, 2006b]. Recent advances in dedicated tools, such as My Online Teacher (MOT) [Cristea et al., 2007] and the Adaptive Courseware Construction Toolkit (ACCT) [Dagger, 2006b, p2], have raised the level of abstraction that the course creator works at when creating adaptive courseware. These tools go some way towards providing a more intuitive interface for constructing adaptive courseware, but do not provide a method for validating the adaptive courseware created. Validation is particularly important in adaptive courseware as it is important to ensure that each individual learner, or defined learner grouping, will be given a pedagogically sound courseware instance that satisfies the course's learning goals.

In addition to the pedagogical challenges that the course creator must deal with, we can observe the following trends in courseware construction:

- Componentisation - Courses are increasingly composed of Learning Objects (LOs) which are small, reusable instructional units typically a lesson, assessment quiz, or possibly a tutorial [Wiley, 2001]. Through reuse the course creator saves time and money, and can use learning resources that have been tried and tested in other courseware. Discovery of LOs has been aided with the advent of the IEEE Learning Object Metadata (LOM) standard [IEEE LTSC, 2002].

- Standards Compliance - There has been a move towards standard and specification compliance in defining courseware, allowing courseware to be moved from one Learning Management System (LMS) to another [Hummel et al., 2004, ADL, 2004]. The formal separation of learning process logic from content in courseware standards and specifications, such as SCORM [ADL, 2004] and IMS LD [Hummel et al., 2004], and the annotation of LOs using a standardised LO metadata, such as IEEE LOM, enables automated courseware validation, as metadata descriptions can be parsed to ensure the LO satisfies some validation criteria.
- Collaborative Courseware Construction - As courseware construction is getting more complex, it is usually now a collaborative, multidisciplinary effort [Ismail, 2001]. Due to this, misunderstandings can easily occur between those involved in courseware construction leading to problems being introduced to the courseware constructed.

These trends have also caused a change in the course creator's role in courseware construction. The course creator now looks to reuse pre-existing LOs in courseware construction instead of producing all learning content needed for courseware. The course creator's role becomes composition-oriented where courseware is composed from existing LOs. Courseware construction is therefore conducted at a higher, more abstract level [Melia and Pahl, 2006a]. Tool support has been developed based on this higher level of abstraction. These tools allow for the production of standard compliant courseware, based on the combination of LOs [RELOAD Project, 2005, Paquette et al., 2006], and the authoring of personalised courseware [Cristea et al., 2003b, Dagger, 2006b]. However, the existing tool support does not provide the course creator with any facility to ensure that the courseware constructed is as originally designed and envisioned by the course creator. The onus is firmly on the course creator to ensure there are no problems in the constructed courseware. With courseware becoming ever more complex, through personalisation, and course creators now developing courseware at a higher level of abstraction, this is an extremely difficult, if not impossible, undertaking for the course creator.

1.2 Research Problem and Challenges

In section 1.1 we identified the need for validating courseware where validation ensures the courseware constructed is as envisioned by the course creator. Our research investigates the automated analysis of courseware allowing for the identification of the following types of courseware problems:

- Scope - Learning goals cannot be achieved with given courseware. The courseware assumes incorrect knowledge for learners.
- Structural - Problems with the composition of courseware and how it is structured.
- Sequencing - Problems with the sequence of how courseware elements will be delivered to the learner.
- Content - Problems with the content used in courseware.
- Pedagogical - Problems with the implementation of a learning approach defined in courseware. Sequencing and content problems can be defined as part of a pedagogical problem.

Courseware validation cannot exist in isolation, it must be part of a holistic courseware construction methodology. It is therefore necessary to investigate the integration of courseware validation into existing courseware construction methodologies.

We have defined our research question as:

> Can courseware requirements in terms of content, structure, sequencing, course scope and pedagogical concerns, which are implicit in courseware creation, be explicitly defined by the course creator and used in the automated validation of a constructed courseware, where validation is integrated with established courseware construction methodologies?

In addressing our research question we can break down the anticipated research challenges as follows:

1. Identify the data available for courseware validation pre-delivery in terms of courseware requirements as defined by those involved in courseware construction.

2. Investigate how the courseware requirements can be represented explicitly.

3. Develop an approach to validate courseware using the courseware requirements. This approach should be optimised towards personalised and personalisable courseware, as it is a major trend in Technology Enhanced Learning (TEL) [Wade and Ashman, 2007].

4. Investigate how courseware validation can be integrated with existing courseware construction tools.

5. Design and implement a proof of concept application that clearly validates our research in terms of its feasibility.

6. Evaluate the research by investigating user acceptance of courseware validation within courseware construction in general and our approach to validation in particular. User acceptance looks at the following:

   - Usability - The validation approach and its tool support must be usable by the course creator.
   - Cost Effectiveness - The validation approach must be cost effective in terms of course creator effort.
   - Effectiveness - The approach captures courseware problems and requirements effectively. The validation approach must be able to validate problems in courseware that the course creator deems to be important.
   - Modifiability - The course creator must be able to customise the validation constraints criteria according to his or her own requirements. Validation support must be flexible enough to be integrated with the TEL specifications used by the course creator in courseware construction.
   - Performance - The performance of our validation approach must be acceptable when compared with the state of the art.

Our research question addresses courseware that is defined using an Educational Modelling Language (EML) [Martínez-Ortiz et al., 2007]. We also assume that the courseware is componentised in that it can be broken up into LOs that are annotated with a view to being repurposed and reused.

Validation is not seen as a replacement for formative [Dick and Carey, 1991] or summative [Carey and Dick, 1991] evaluation of courseware, but as a complementary activity. Validation does not address complex pedagogical issues such as the clarity of instruction, LO quality and LO's impact on learners or learner motivation issues.

1.3 Solution and Evaluation

Our solution for addressing the research question and challenges outlined in the previous section is to explicitly model the courseware design in terms of its requirements. This courseware model is then constrained using a model constraint language. To do this our approach defines a Domain Specific Modelling Language (DSML) to capture courseware requirements and uses a constraint language to allow the course creator define what is valid and invalid courseware.

Our approach consists of five components, as outlined below:

- A Validation Framework - Definition of what can be validated in courseware at the pre-delivery/post-construction stage of the courseware life-cycle and the courseware requirements to be used in validation.
- Domain Specific Modelling Language (DSML) - Definition of a modelling language for capturing courseware, courseware content and the constructed courseware requirements. We have chosen to develop a new DSML for the purposes of validating courseware for the following reasons:
  - There is currently no formalised way to capture the courseware requirements at the pre-delivery/post-construction stage of courseware construction.
  - Courseware validation is an activity that must be integrated with courseware

construction. The specifications used in courseware construction are in flux as they are being standardised. Our approach limits the effects of this standardisation process by defining an independent courseware definition and interoperating with the state of the art.

- Validation Constraint Language - Definition of a constraint language that can capture what is valid and invalid courseware.
- Courseware Validation Process - Outline of how the course creator validates courseware using our approach.
- Proof of Concept Implementation - Courseware construction tool that allows for validation, exemplifying how validation can be integrated into courseware construction.

The validation framework sets the scope of our research, by defining the courseware construction concerns. These are courseware requirements that must be addressed in courseware construction. The courseware construction concerns represent the information available for courseware validation at the pre-delivery/post-construction stage of the courseware life-cycle, when validation takes place. The validation framework also outlines the types of courseware validation our research addresses.

The purpose of the DSML is to capture all the data available at the pre-delivery/post-construction stage of the courseware life-cycle. This data is the courseware that has been constructed, the content used in courseware and the courseware construction concerns that were used in defining the new courseware.

Courseware validation criteria are formulated using a model constraints language. The model constraints language is used to constrain what is an allowable courseware definition. A valid courseware is one that satisfies these modelling constraints. Model constraints can be defined using the courseware construction concerns defined in the DSML.

We define a courseware validation process as a set of activities that the course creator undertakes to validate courseware. This process involves firstly defining all data available at the pre-delivery/post-construction stage of courseware construction using the DSML, then defining the courseware validation constraints, after which the courseware is checked to ensure it adheres to the constraints defined.

Courseware validation must be integrated with the state of the art in courseware construction. Interoperability with courseware construction tools and platforms is achieved through model transformation technology (section 2.5). We illustrate this in our proof of concept implementation.

The proof of concept implementation is also used to validate the feasibility of our courseware validation approach.

We evaluate our approach to courseware validation in the following ways:

- An evaluation of the suitability of our DSML is carried out by analytically comparing the expressivity of the DSML defined in this thesis with languages that represent similar data from the state of the art.

- A software implementation is used in user trials to assess user acceptance of a model-driven approach to courseware construction that leads to validation. This also serves to assess the user acceptance of the DSML we have defined for courseware validation.

- We investigate how our approach compared with state of the art courseware validation approaches in terms of performance and modifiability. We also consider how easy it would be for the course creator to modify the validation criteria, the courseware specification or the knowledge infrastructure used in validation. In looking at performance we compare our novel approach to courseware validation with the validation approach used in the state of the art generally. This evaluation centres around a discussion on the difference between constraint-based validation, used in our solution, and learner simulation based validation, which is prevalent in the state of the art.

1.4 Contributions

In this section we highlight the contributions the research documented in this thesis makes to the state of the art in TEL.

- This research looks at courseware validation at the pre-delivery/post-construction stage of the courseware life-cycle. This stage of the courseware life-cycle has only limited data to use in validation, most notable is that there is no learner experience data to use. In this thesis we identify the "courseware construction concerns". The courseware construction concerns are the complete data about a given courseware at the pre-delivery/post-construction stage of courseware construction. This data is identified through systematic analysis of the pre-delivery/post-construction stage of courseware construction. To our knowledge this is the first time the data available for courseware validation at the pre-delivery/post-construction stage of courseware construction has been defined.

- We formally define a new DSML for courseware validation at the pre-delivery/post-construction stage of the courseware life-cycle. This DSML is known as the Courseware Authoring Validation Information Architecture (CAVIAr). CAVIAr is a purpose built DSML for courseware validation. It allows for the formal definition of the courseware construction concerns for the purpose of courseware validation.

- A novel constraint-based approach to validating a courseware definition is defined that focuses on validating the courseware structure and composition rather than validating the possible learner paths learners can take in courseware. This validation approach is courseware composition-oriented, by examining structural definition of courseware. This is different to the state of the art which, in general, looks to validate courseware by simulating learner progression through courseware. Composition-oriented validation is more suited to personalised courseware due to the variance in it.

- A model-driven validation process is defined, outlining the activities that the course creator must perform for validation. Models provide an intuitive way for the course creator to define the courseware construction concerns through CAVIAr. Through these activities we also outline our approach to interoperability with the state of the art in TEL and knowledge representation specifications and standards through model transformation technology.

We also describe how model transformation technology can be used to provide for interoperability in TEL. We also provide details of a software implementation as a proof of concept, demonstrating the feasibility of our validation approach.

The contributions detailed in this thesis have also appeared in numerous peer reviewed publications the most significant of which are:

- Melia, M. and Pahl, C. (2009) *Constraint-based Validation of Adaptive e-Learning Courseware*. IEEE Transactions on Learning Technology (IEEE TLT) 2(1), pp 37-49.

- Melia, M. and Pahl, C. (2008) *Towards the Validation of Adaptive Educational Hypermedia using CAVIAr*. The Sixth International Workshop on Authoring of Adaptive and Adaptable Hypermedia A3H2008 AH2008 Workshop Proceedings Hannover, Germany.

- Melia, M. and Pahl, C. (2007) *Pedagogical Validation of Courseware*. The Second European Conference on Technology Enhanced Learning (EC-TEL 2007). Springer-Verlag, LNCS Series. Crete, Greece.

- Melia, M. and Pahl, C. (2007) *An Information Architecture for Validating Courseware*. The First International Workshop on Learning Object Discovery and Exchange (LODE2007) at EC-TEL'07. CEUR Workshop Proceedings, ISSN 1613-0073 Crete, Greece.

- Melia, M. and Pahl, C. (2006) *Semantically-enabled Model Driven Course Composition*. The First European Conference on Technology Enhanced Learning (EC-TEL 2006) - Doctorial Consortium Session. Crete Greece.

1.5 Thesis Outline

In this section we outline the structure of this thesis.

Chapter 2 presents a review of technologies, approaches and terminology in TEL. The chapter also presents an overview of software modelling technologies, as these technologies are central to our solution.

In chapter 3 we provide context for our research by presenting the state of the art in courseware authoring tools and approaches and then present the state of the art in courseware validation research. In concluding this chapter we present a comparison framework, which summarises our review of the state of the art.

Our research contribution begins in chapter 4, where we define a conceptual framework for validation, outlining the overall scope for courseware validation in our research.

In chapter 5, we present the Courseware Authoring Validation Information Architecture (CAVIAr), a DSML that allows the course creator to model the courseware construction concerns and the constructed courseware in the context of these concerns.

To validate the constructed courseware, CAVIAr includes a constraint-based validation model, where validation rules are defined on the constructed courseware using the courseware construction concerns. We describe the CAVIAr Validation Model in chapter 6, outlining the constraint types that can be defined using CAVIAr.

We present the courseware validation activities and define a courseware validation process in chapter 7. This section also provides an example case-study whereby a course creator has created courseware for undergraduate computing students on the basics of databases. We describe how the course creator validates this courseware using CAVIAr.

A proof of concept implementation of CAVIAr is outlined in chapter 8. This implementation is known as MIKAEL (Management Infrastructure for Knowledge-based Adaptive E-Learning) . MIKAEL allows the course creator to construct courseware based on defining the CAVIAr courseware construction concerns. Once the courseware has been developed MIKAEL also provides the course creator with courseware validation functionality. MIKAEL is specification and standard agnostic. We outline how MIKAEL can easily be integrated with current and upcoming standards and specifications in TEL and related areas, such as knowledge management and the Semantic Web [Daconta et al., 2003].

We evaluate our work in chapter 9. The chapter evaluates the user acceptance of CAVIAr and CAVIAr-based courseware validation, as defined in our research challenges in section 1.2.

In chapter 10, we summarise our research, present our final conclusions and outline future work in this area.

Chapter 2

Background

2.1 Introduction

In this thesis we will outline how courseware can be validated. Our validation approach is based on the definition of a Domain Specific Modelling Language (DSML), to capture the courseware conceptual design and its requirements, with a constraints language that constrains the allowable courseware configurations.

In this chapter we firstly describe what courseware is and how courseware is constructed (section 2.2). We then outline software modelling technologies used in our courseware validation approach. Section 2.3 gives an overview of the classical approaches to conceptual modelling in software engineering and section 2.4 describes the Object Constraint Language (OCL), a complimentary constraint language for software models used to eliminate ambiguity. Section 2.5 looks at Model Driven Engineering (MDE), a software engineering approach based on modelling. We conclude this chapter by summarising the main points made and outlining the connection between courseware design and software models.

2.2 Courseware

For the purposes of our study, we define a course as a planned body of instruction with a recognised start-point(s) and end-point(s), which is delivered over a specified time period.

We define courseware as an explicitly defined, machine-readable course definition. Courseware uses software to deliver Technology Enhanced Learning (TEL) content according to some defined instructional design.

TEL content is interactive and non-interactive learning content delivered to learners in a software environment that aims to enhance the learner's learning experience. An instructional design specifies what TEL content is delivered to an individual learner and when it is delivered. An instructional design is an instantiation of a more general-purpose instructional design theory. Instructional design definitions in courseware can allow for dynamic courseware, where the courseware decides at delivery time what TEL content to deliver and how it will be delivered

Based on the literature, we have identified six key levels of courseware granularity [NQAI, 2003, Bajnai and Stienberger, 2003, Jovanović et al., 2006a]. These are as follows:

- Qualification Programme - This is typically a very large course leading to a qualification as defined in the National Framework for Qualifications (NFQ) in Ireland.
- Course - This is a large course, which takes over 15 hours to complete. It is typically one 12-13 week course covered in a university semester as part of a qualification programme.
- Module - Many modules make up a course. Each module in a course covers one of the core concepts in the course. A module can be broken down into lessons.
- Lesson - A lesson is generally taught to the learner in a single sitting. A lesson generally teaches the learner about a single concept. A lesson can be broken down into LOs.
- Learning Object (LO) - A LO is interactive and non-interactive TEL content that aims to bring the learner to a learning goal. LOs can be broken down into content units.
- Content unit - LOs are made up of content units, which are typically a picture or a piece of text.

It is important to note that the courseware granularity levels stated above are not commonly accepted but are exclusively used in this thesis to allow for clarity when discussing courseware granularity levels.

2.2.1 Courseware Scope

The courseware scope defines what a course covers. It specifies the course start-point and end-point. The courseware start-point is defined in terms of the learner's assumed initial knowledge of a subject domain, while the courseware end-point is defined as a point in the courseware when the learner can finish the course and the learner's learning goals are satisfied. The courseware scope is therefore defined in terms of knowledge. The courseware structure defines how a learner can get from the courseware start-point to the courseware end-point.

For the purposes of our research we follow a pragmatic rather than formalised approach to defining knowledge, based on instructional design literature [Albert and Stefanutti, 2003]. In our research we make a distinction in the types of knowledge imparted through courseware, conceptual knowledge and skills knowledge. Conceptual knowledge is typically attained though classroom or didactic learning, while skills knowledge is typically attained though practical training [Kenny, 2006]. Conceptual knowledge learning can be mapped to "verbal information" in Gagné's learning outcomes, while skills knowledge can be mapped to "intellectual skills" [Gagné et al., 2005]. A knowledge level can also be defined as a numeric value between 0 and 1, where 0 represents no knowledge and where 1 represents full knowledge [Melia and Pahl, 2009].

2.2.1.1 Learner's Learning Goals

The learner's learning goals define the desired post courseware delivery knowledge state for each learner. Learning goals are therefore defined in terms of knowledge. The course learning goals are the starting point of course development, on which assessment and instructional design are based [Dagger, 2006b, ch. 3].

In order to accurately assess whether or not courseware satisfies each course learning goal, a formative evaluation (post-delivery) can be carried out to insure a learner will attain the desired knowledge [Dick and Carey, 1991].

At the pre-delivery stage of courseware construction we can assess if the scope of the courseware is sufficient to satisfy all learner's learning goals and that the type of TEL content used is suitable for the desired type and level of knowledge.

2.2.1.2 Learner's Assumed Initial Knowledge

Assumed initial knowledge is a statement of the knowledge we expect learners to begin a given courseware with. Assumed initial knowledge can also be expressed in terms of conceptual and skills knowledge. The aim of a course is to define instruction that will take the learner from assumed initial knowledge to the learner's learning goal. Capturing assumed knowledge in courseware validation is crucial, as courseware instruction should build on existing learner knowledge in order for the learner to achieve a learning goal.

2.2.2 Courseware Structure and Sequencing

The courseware structure facilitates instruction by bringing each learner from some assumed initial knowledge state to the learning goal. The courseware structure defines how the courseware is brought together but also normally has sequencing definitions embedded in the structure. For this reason, we look at the courseware structure and sequencing together in this section.

In looking at courseware structure a distinction can be made between the micro and macro (strategy) levels as defined by Reigeluth [Reigeluth, 1983b]. The micro level is concerned with how a single idea or concept is taught, e.g. addressing the sequencing of LOs that address the same concept. The macro level addresses course concerns at the concept level, e.g. what concepts are learnt by the learner and in what order?

2.2.2.1 Micro-Level Courseware Structure

Micro-level courseware structure is concerned with the instructional design used in teaching a core courseware concept. It is at the micro-level that the course creator makes decisions on the best instructional design for teaching a concept and ensuring that the instructional design used in the courseware accurately implements that strategy. An example of an an approach the looks in detail at intra-conceptual sequencing of learning content can be found in [Ullrich, 2005], where Ullrich investigates how Hierarchical Task Network (HTN) planning can be used to define an instructional design to teach a given concept.

The course creator may define a default sequence of learning events, where each event must take place for a concept to be taught. For this purpose, Gagné et al. outline the "Events of Instruction", a series of instructional events which must take place for micro-level learning to take place [Gagné et al., 2005]. These instructional events are; gaining attention, informing the learner of lesson objective, stimulating recall of prior learning, presenting stimuli with distinctive features, guiding learning, eliciting performance, providing informative feedback, assessing performance and enhancing retention and learning transfer. Gagné et al. believe that all events are necessary in roughly this order. The exclusion of any event must be a deliberate and conscious decision of the course creator based on the learning audience, the learning task or both.

2.2.2.2 Macro-Level Courseware Structure

The macro-level courseware structure is concerned with how the teaching of concepts is sequenced in courseware. In defining the courseware structure the course creator attempts to define courseware sequencing that brings the learner from an assumed initial knowledge state, to a learning goal, in the most effective and/or efficient manner.

Learning takes place in the context of other knowledge. Knowledge that is required for learning is known as pre-requisite knowledge. Gagné et al. describes pre-requisite knowledge as "critical for the rapid, smooth learning of [a] new skill", the absence of which would make learning the new skill impossible [Gagné et al., 2005, p184]. In traditional courses, sequencing is dictated by the instructor. In courseware the learner becomes a more active player, and can influence the delivery sequence of course material. This can be done knowingly or unknowingly. The course creator can still specify a sequence but can also empower the learner to choose their own sequence. This extra dimension to courseware sequencing makes courseware structure validation very important, where validation must ensure that any topic sequence that is available to the learner is instructionally sound.

2.2.3 Learning Content in Courseware

As mentioned in section 1.1, Learning Objects, used in courseware, can be annotated using the IEEE Learning Object Metadata (LOM) standard. LOM is used to “facilitate search, evaluation, acquisition and use of learning objects” [IEEE LTSC, 2002]. The standard is quite large incorporating many characteristics of LOs. Due to its coverage the LOM standard can be used to validate LOs used in courseware. There are seven sections of the LOM standard, these are:

- General - General cataloguing information such as the LO name, language, keywords, natural language description, structure, aggregation level and a unique identifier.
- Life Cycle - Information about versioning, status and who has contributed to the LO.
- Meta-Metadata - Information about the metadata such as who is cataloguing the LO and the language of the metadata.
- Technical - Information such as the format of the LO, the file size, file location and system requirements.
- Educational - Metadata describing pedagogical details, such as the difficulty, the interactivity type, learning resource type and semantic density of the LO.
- Rights - Details of the LO cost and copyright.
- Relation - How this LO is related to other LOs. The relation can express a range of relation types such as *basedOn, isPartOf, isVersionOf, isFormatOf, references* and *requires*.
- Annotation - Comments on how, by whom and when people are using this LO. This allows for educators to supply some feedback on the LO.

- Classification - This section of the LOM allows the LO to be classified using some specific external classification system. The person annotating the LO must provide details on what classification system is being used, where this LO falls in the classification system and what the classification means.

The range (allowable values) for the majority of the LOM attributes is free text (e.g. description in the *general* section), while some only allow for a set of restricted values (e.g. structure in the *general* section allow only the following values: atomic, collection, networked, hierarchical and linear). LOM attributes can also be set to reference some external classification system, or reference external resources such as other LOs. An XML schema can also be used within LOM to enhance meaning of LOM values. For example, when annotation refers to a person the *vCard* standard can be used [Dawson and Howes, 1998]. The course creator can use this metadata to define validation concerns. We will look at this in more detail in section 4.4.3.

2.2.4 Pedagogical Strategy

A pedagogical strategy sets out a strategy for learning. In courseware we are limited in the extent to which pedagogical strategy can be realised. We confine it to defining what LOs are delivered to which learners and when. Pedagogical strategy overlaps with many of the other courseware construction concerns, such as the courseware sequencing, as both are concerned with how the courseware is delivered to learners.

Courseware pedagogical strategy allows courseware to adapt to the needs of individual learners. The learner is the most complex concern in course construction as the characteristics of a learner are infinite. When representing a learner we generally restrict ourselves to representing key characteristics that dictate the way the learner learns. To limit the complexity the learner introduces to the courseware definition, the course creator can use learner stereotypes where a stereotype groups learners according to some common attribute(s) (e.g. software engineering students as opposed to electrical engineering students) [Kay, 2000].

Adaptive behaviour in adaptive courseware is defined using learner characteristics. Dagger outlines the learner characteristics that courseware can adapt to as; prior knowledge and competence, learner aims and goals, learner preferences, learner courseware interaction history, cultural background, cognitive and learning style, preferred communication style and needs and the learner delivery environment [Dagger, 2006b].

2.2.5 Courseware Quality

The quality of courseware can be assessed by evaluating particular attributes of the courseware, known as quality attributes. The validity of a given courseware is judged on the combination of these attributes.

Grützner et al. identify four central quality attributes to which courseware validity can be judged, these are [Grützner et al., 2004]:

- The Content of Learning Materials - What content is taught to the learner, the format the content takes and its suitability.
- The Presentation of Learning Materials - Addresses how courseware materials are presented to the learner, addressing courseware sequencing and courseware structural issues.
- How Learning Materials are Taught (i.e. pedagogic content) - This dimension of quality addresses the pedagogical approach to content delivery.
- How Well Courseware Engages - Looks at the level of learner engagement with the courseware.

In validating courseware, we aim to pinpoint problems in courseware by identifying courseware elements that violate quality attributes. The quality attributes outlined above offer general guidelines for evaluating courseware.

Various educational bodies also specify guidelines for evaluating courseware, using a variety of quality attributes. The Texas Education Agency has defined a comprehensive evaluation matrix for evaluating courseware [Blackerby et al., 2002]. The quality attributes used in this matrix are:

- Course Design - Ensures the learner's learning goals are addressed. This looks at the syllabus used to address the learning goals and also at how well the courseware engages the learner.
- Course Content - Looks at the how course content is organised, and if the course content provided is sufficient to achieve each learner's learning goals.
- Instructional Strategies and Activities - Does the instructional strategy suit the course learning outcomes and learners?
- Learning Community - The course affords the opportunity to create a learning community and encourages collaborative methods.
- Student Assessments - Can the course effectively assess learner's knowledge at any given point in the courseware.
- Technology Integration - Technology used enhances the learner's learning experience, technology and media integration is seamless and the potential of technology failure is addressed through the instructional design and alternative delivery methods.
- Course Effectiveness - Addresses the summative evaluation concerns of the courseware.

Other efforts in the area of courseware quality have concentrated on establishing the quality of learning content in courseware. One such effort is the Learning Object Review Instrument (LORI), which provides users of a given LO a common framework with which to evaluate the LO [Nesbit et al., 2003].

The quality attributes presented in the literature specify general courseware quality attributes. For the purposes of courseware validation we focus on those quality attributes that can be evaluated before the course is delivered to the learners, known as the "courseware construction concerns". We define these concerns in section 4.3. The courseware construction concerns are focused, measurable qualities of courseware and can also be assessed prior to courseware delivery, allowing for correction before delivery to learners.

2.2.6 Courseware Standardisation Efforts

Standardisation efforts in courseware have allowed for the de-coupling of TEL content from instructional logic, where instructional logic is the explicit machine-readable definition of an instructional design. Many approaches to defining instructional logic can be found in the literature in the form of Educational Modelling Languages (EMLs) [Martínez-Ortiz et al., 2007]. Two of these approaches are recognised IMS specifications, IMS Simple Sequencing [IMS, 2003c] and IMS Learning Design (LD) [IMS, 2003b]. More specialised EMLs have also be defined for specific types of e-learning environments, for example the LAOS language, developed by Cristea & de Mooij, allows the course creator to define Adaptive Educational Hypermedia (AEH) [Cristea and de Mooij, 2003]. AEH is described in more detail in section 3.2.2.

Recently, LOs have grown in popularity due to their advantages in courseware construction, such as a reduction in courseware construction time and effort, and also allowing course creators to reuse tried and tested TEL content. Specifications have also been developed that allow for the packaging of LOs, such as IMS Content Packaging (CP) [IMS, 2003a].

Learning Management Systems (LMSs), such as Moodle[1], Blackboard[2], and Sakai[3], provide a delivery platform for courseware delivery. LMSs can use the courseware specifications to import and export courseware from one LMS to another. This allows for courseware to be developed independent of the eventual courseware delivery platform. The main courseware specifications used in the import and export of courseware are the ADL SCORM [ADL, 2004] and the IMS Learning Design (LD) [IMS, 2003b] specifications. As the vast majority of courseware is delivered through a LMS, in this section we look in more detail at the specification efforts that allow for interoperability with LMSs.

[1]`http://www.moodle.org`
[2]`http://www.blackboard.com`
[3]`http://www.sakaiproject.org`

2.2.6.1 SCORM

The Shareable Content Object Reference Model (SCORM), is a specification that defines collection of standards and specifications used to describe courseware for the purposes of courseware portability. The most recent published SCORM specification is the SCORM 2004 specification [ADL, 2004]. This specification is made up of three parts, SCORM Content and Aggregation Model (CAM), SCORM Sequencing and Navigation (SN) and SCORM Run-Time Environment (RTE). The previous version of this specification was the SCORM 1.2 specification which did not have the sequencing and navigation definitions.

The SCORM CAM, allows for the packaging and description of a given courseware. The main specification used to define the SCORM CAM is the IMS Content Packaging specification [IMS, 2003a]. Metadata standards, such as the IEEE LOM standard, can be embedded into the SCORM CAM to describe courseware.

The SCORM SN, allows for the definition of how learning content in SCORM is sequenced for the learner. The principle specification used in SCORM SN is the IMS Simple Sequencing (SS) specification [IMS, 2003c]. The SCORM SN is defined in two main ways, through “sequencing control modes” and “sequencing rules”. Sequencing control modes define how a learner can interact with a sequencing “cluster” where a cluster is an aggregation of learning content with an aggregation level of one. We have outlined sequencing clusters in the diagram in figure 2.1. Sequencing rules allow the course creator to define condition-action rules for SCORM courseware.

The SCORM RTE allows for communication between a SCORM package and a LMS. The principle standard used for this is the IEEE ECMAScript Application Programming Interface for Content to Runtime Services Communication [ADL, 2004].

2.2.6.2 IMS Learning Design (LD)

The IMS LD specification aims to capture learning (instructional) designs to be deployed for a specific learning situation [Koper, 2005]. The IMS LD specification started as the Educational Modelling Language used at the Open University of the Netherlands (OUNL), but

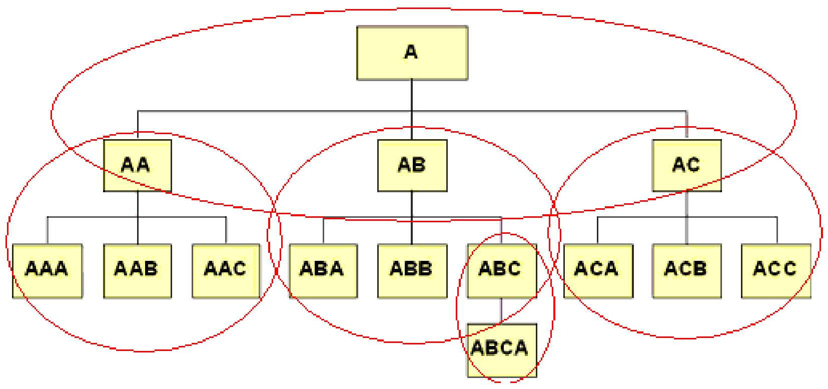


Figure 2.1: Sequencing clusters as defined in IMS Simple Sequencing [IMS, 2003c]

has since been brought under the IMS umbrella, integrated with the other IMS specifications and renamed IMS Learning Design (LD).

IMS LD is based on a theatre script metaphor. A learning design is made up of a set of *plays*, which in turn contain *acts*. An *act* has a set of *role-parts*. There are two types of roles in a learning design, *staff* roles and *learner* roles. The learning design workflow is managed by the *method*, which is designed towards a set of *prerequisites* and *learning objectives*. A *role-part* in IMS LD links an *activity* to a *role* [Olivier and Tattersall, 2005]. An *activity* defines a learning *environment*, which is a set of *learning objects* and *services*.

Learning design has been designed in levels, allowing the course creator to increase the complexity of the learning design they are using incrementally. These levels are as follows [Olivier and Tattersall, 2005]:

- Level A - The core of the LD language, allows for the description and packaging of courseware. Everything described thus far has been level A.
- Level B - Allows sophisticated adaptive learning specification using properties and conditions.
- Level C - Allows for notifications.

Level B properties and conditions allow for the capturing of information about the learners, and the state of the learning design itself. Conditions can be then defined on these properties. Level C provides for greater interactivity and control over LD during delivery, allowing for a form of event-driven messaging in the learning design.

Figure 2.2 illustrates the conceptual structure of the IMS LD specification.

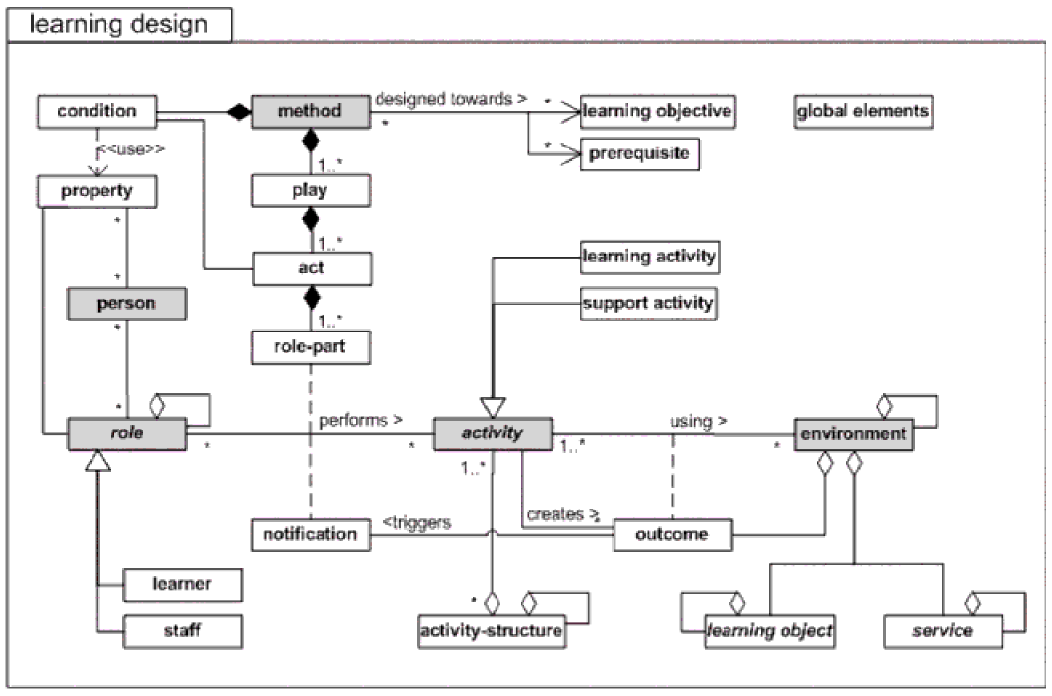


Figure 2.2: IMS LD Information Model [IMS, 2003b]

2.3 Software Modelling Technologies

The courseware validation solution, defined in this thesis, is based on the definition of a DSML, using conceptual modelling technology. In [Daconta et al., 2003, ch.7], Daconta et al. outline a series of ontology types with varying levels of expressivity, known as the ontology spectrum. To define a DSML for courseware validation we require an ontology expressivity level that can define a fixed vocabulary. We have determined that the level of expressivity needed for our research is that of a "conceptual model" in the ontology spectrum [Daconta et al., 2003] for the following reasons:

- We expect course creators to use the DSML in courseware construction for this reason it must be as intuitive as possible and therefore limit its complexity.

- The DSML is expected to capture courseware and be interoperable with courseware specifications. To allow for this the expressivity level of the DSML defined in this thesis must be the same as or less than that used to define the state of the art. The state of the art is defined using conceptual models.

- In order to capture courseware requirements defined in terms of knowledge the DSML must be defined using an expressivity level the same as or less than that of those standards and specifications used to define knowledge. Knowledge standards are generally defined using the expressivity of a conceptual model or higher.

To this effect we examine software modelling technologies used to capture conceptual models, these being the Meta Object Facility (MOF) [OMG, 2003a] and the Unified Modelling Language (UML) [Eriksson et al., 2003]. Both MOF and UML are Object Modelling Group (OMG) standards. The OMG is an industry consortium that "maintains computer-industry specifications for interoperable applications" [Eriksson et al., 2003]. We will also look at the Eclipse Modelling Framework (EMF) [Steinburg et al., 2008]. EMF provides a modelling language that allows for a generative approach to software development within the Eclipse IDE [Czarnecki and Eisenecker, 2000]. MOF and EMF modelling constructs are similar to UML modelling constructs and as UML is a widely known modelling language we will only briefly introduce MOF and EMF.

Software models are traditionally used as a design artefact in software development. Improvements in their semantics have allowed for them to become actual development artefacts, capable of generating code. The use of software models as development artefacts in software engineering is known as Model Driven Engineering (MDE) [Schmidt, 2006]. In this thesis, we will outline how MDE can be used with our DSML in courseware construction, and can also be used to allow for interoperability between the DSML and TEL. MDE is described later in this chapter in section 2.5.

Software modelling technologies are the basis for the courseware validation approach defined in this thesis. The software modelling technologies described in this section allow for the definition of a Domain Specific Modelling Language (DSML) in the “metamodelling technical space” [Djurić et al., 2006]. When this DSML is used with the Object Constraint Language (OCL) it allows for the validation of courseware by constraining the allowable courseware definitions. As OCL is not widely known, we therefore define OCL in a tutorial like format, in section 2.4.

2.3.1 Meta Object Facility (MOF)

The MOF is a universal mechanism for defining modelling constructs. The MOF itself is self-defined. A MOF modelling architecture has a four layer modelling stack for defining modelling languages. We outline the MOF modelling stack in table 2.1.

Table 2.1: MOF modelling stack

Metalevel	Description
M3	The meta-metamodel level, defined using MOF
M2	The metamodel level defined using MOF. The M2 layer used to define modelling languages e.g. UML metamodel
M1	The model level defined using modelling elements defined in a metamodel at the M2 layer e.g. UML model
M0	The actual runtime instance of the software model defined at M1

2.3.2 Unified Modelling Language (UML)

The Unified Modelling Language (UML) is used to describe software and software requirements. The UML metamodel is defined by the OMG using MOF [OMG, 2007]. There are two types of UML models, structural and behavioural. Structural models define how a system is composed, while a behavioural model documents the dynamic behaviour of a system. In table 2.1 we have also noted the modelling levels that the UML metamodel (UML definition) and a UML model is defined at in the MOF modelling stack.

2.3.3 Eclipse Modelling Framework (EMF)

EMF is an open source modelling framework which integrates into the Eclipse platform [Steinburg et al., 2008]. The primary purpose of EMF is to allow for generative programming within the Eclipse IDE. Developers can define software using an ECore model, a modelling language for EMF. ECore models can then in turn be used to generate Java code. ECore is closely aligned with the Essential MOF (EMOF), a subset of the OMG MOF specification. An ECore model can be generated from an XML schema, a set of annotated Java interfaces or an UML2 model. To allow for the generation of Java code from an ECore model EMF uses a model-to-text generator such as the Java Emitter Template (JET). We will describe JET in detail in section 8.4.3.

Many powerful developer applications have been built on top of EMF. One such tool is the Graphical Modelling Framework (GMF). GMF is a framework for designing modelling tools based on EMF models. To create a GMF modelling tool an EMF model is defined as the tool's metamodel. Model diagrams developed in the GMF application must then conform to this metamodel. We describe GMF in detail in section 8.4.1.

2.4 The Object Constraints Language (OCL)

In this section we take an in-depth look at OCL, its syntax, semantics and general use. We will concentrate on the parts of the OCL most applicable to our work. A more detailed description of the OCL can be found in [OMG, 2003b] and [Warmer and Kleppe, 2003].

OCL is a formal constraint language originally designed to describe expressions to enhance the semantics of UML models. OCL typically takes the form of invariant constraints, defined on modelling constructs in a UML model. An invariant is a rule that must be true when the model is instantiated. OCL can also be used to query a UML and ECore model.

OCL is a declarative language that has no side effects on the model it is defined on.

This means OCL cannot change the information in the UML model, it can only add to the semantics of the model by constraining the allowable model instances.

The OCL standard, specified by the OMG, states that OCL should be used for the following purposes [OMG, 2003b]:

- “As a model query language.
- To specify invariants on classes and the types allowed in the class model.
- To specify type invariant for stereotypes.
- To describe pre- and post-conditions on operations and methods.
- To describe guards in activity diagrams.
- To specify target (sets) for messages and actions.
- To specify constraints on operations.
- To specify derivation rules for attributes for any expression over a UML model.”

In this section we will provide an overview on defining OCL constraints, we will also define OCL’s application areas and assess the tool support available for defining OCL.

2.4.1 OCL Language Constructs

In this subsection we outline the major OCL language constructs used to define constraints and queries in OCL. The main reference points used are the OMG standard [OMG, 2003b], the text by Warmer and Kleppe [Warmer and Kleppe, 2003], and Richters Ph.D. thesis [Richters, 2001].

2.4.1.1 Expressions

OCL expressions are based on set theory and predicate logic. Formal mathematical semantics for OCL are defined in [Richters, 2001]. OCL does not use any mathematical notation, as one of the design motivations for the standard was the need for the rigour and precision of mathematics with the ease of natural language. The result is a precise unambiguous language that is easily written and read by Object Oriented practitioners [Warmer and Kleppe, 2003, p17].

Literals and variables can be used to build simple expressions in OCL. More complex OCL expressions can be defined, such as conditional branches, by using OCL's if-then-else notation.

The value of an object property, defined in a class diagram, is specified using the dot notation in OCL, i.e. *classname.propertyname*. The dot notation allows the user to build paths around a given model. To navigate to an associated class the association end role of the associated class is used as the property name. Properties on collections are assessed using an arrow "->" followed by the name of a OCL collection operation (see section 2.4.1.8).

2.4.1.2 Types

The most basic OCL types are; Boolean, Integer, Real and String. These types can be manipulated using logic operations (Boolean), arithmetic operations (Integer and Real) and string manipulation operations such as substring (String).

Three specialised types of collection classes are also defined in OCL; Set, Bag, and Sequence. A set is a collection of objects, which does not allow for duplicates, Bag allows for duplicates and a Sequence is an ordered set. Manipulation of collections is done using various collection operations. We look at the OCL collection operations in detail in section 2.4.1.8.

2.4.1.3 Context

All OCL expressions have a grounding in a model, which a given OCL constraint is defined in terms of, this is known as the constraint context. In listing 2.1 the constraint context is the class *Car*, this means that the OCL constraints are defined from this model construct. Here *numwheels* is an attribute of the *Car* class. The invariant states that the *numwheels* attribute must be equal to four for the invariant to be satisfied.

Listing 2.1: Defining an additional operation on the Car class

```
context Car
    inv wheels: numwheels = 4
```

2.4.1.4 Invariants

Invariants in OCL are defined with the keyword *inv*. An invariant must be true for an instance of a model. It provides a method to constrain the forms objects can take in the instance space. In listing 2.1 we have demonstrated an invariant stating that a car must have four wheels.

Invariants may have an optional name, which is defined after the *inv* keyword and before the colon indicating the start of the invariant. In listing 2.1 the invariant is named, *wheels*.

2.4.1.5 Pre-conditions and Post-conditions

Pre-condition states and post-condition states can be defined on class operations using OCL. The precondition constrains what system state the context class operation can be invoked in while the post-condition states what the system state must be once the operation has been invoked.

In listing 2.2 we have outlined an example use of a pre-condition and a post-condition for the *start()* operation of the Car class. The Car class's *start()* operation is defined as the context. To define the operation as the context, we specify the context class and then after a double colon ("::"), we state the operation. The double colon indicates that the operation call is an instance level event - a system state. The constraint states that the *start()* operation must only be invoked when the car's engine is off (pre-condition) and the engine must be on after invoking the operation (post-condition).

Listing 2.2: Defining an additional operation on the Car class

```
context Car :: start() : Boolean
pre: engine.on = false
post: engine.on = true
```

2.4.1.6 Let

The *let* keyword in OCL allows for the definition of local variables for an invariant and is defined as a sub-expression. In the example in listing 2.3, an initial local variable is defined for the invariant using the let keyword. The local variable, *old* defines what an "old" car is (i.e. a car more than ten years old), the invariant, *service* is then defined using this local variable after the keyword *in*.

Listing 2.3: Defining an additional operation on the Car class

```
context Car
inv service: let old: Boolean =
    age > 10
    in
    if old = true then
        serviceRequired = true
    else
        serviceRequired = false
    endif
```

2.4.1.7 Def

The OCL *def* keyword allows for the definition of extra object attributes and operations using OCL. These operations and attributes can then be used by other OCL constraints. In listing 2.4, we use the *def* keyword to define two operations. The first operation is *getNumPassengers()* on the *Car* class, which evaluates the number of passengers in a *Car* instance. The second operation *getPassengerName(x:Integer)* gets the name of the passenger at the index *x*. This operation illustrates how parameters can be passed into an OCL operation.

Listing 2.4: Defining an additional operation on the Car class

```
context Car
    def getNumPassengers() : Integer = self.passengers->size()
    def getPassengerName(x:Integer) : String= self.passengers->at(x)
```

2.4.1.8 Collection Operations

OCL offers a variety of collection operations, which aid the user in the manipulation and querying of OCL collections, these include:

- *sum()* - Returns the sum of all elements in a collection.
- *size()* - Returns the number of elements in a collection.
- *isEmpty()* - Returns true if the collection has zero elements.
- *notEmpty()* - Returns true if the collection has more than zero elements.
- *select(expr)* - Returns elements in a collection where *expr* is true.
- *collect(expr)* - Returns collection which results from evaluating *expr*.
- *forAll(expr)* - *Expr* must be true for all elements in the collection.

2.4.2 OCL Applications

As we have mentioned the main use of OCL is to enhance the expressiveness of UML models, by adding additional information to the model that cannot be expressed using UML's visual notation. Since its original standardisation in 1997 OCL has evolved, and its applications have grown. OCL is now used in the following application areas:

- The largest use of OCL remains refining UML model definitions. The importance of OCL in this regard is especially important with the advent of Model Driven Engineering (MDE) in software development [Warmer and Kleppe, 2003, ch1].
- OCL is also being used to refine DSMLs defined in the metamodelling technical space using meta-metamodel languages such as MOF or Ecore [Gronback, 2009]. To do this OCL is used to constrain metamodel definitions.
- OCL is used to define some model transformation languages such as the Atlas Transformation Language (ATL) [Jouault and Kurtev, 2005].

Tool support can be provided to the OCL developer in a number of ways. Hussmann et al. outline the most important aspects of tool support needed in order to ensure the extra effort needed to define OCL constraints is cost-effective [Hussmann et al., 2000]. In their work they outline key features that an OCL tool should have:

- Syntactic Analysis - Parsing OCL expression for syntactic errors.
- Type-checking - Enables automatic static type checking of OCL.
- Logical Consistency checking - Ensure that OCL constraints are not contradictory.
- Dynamic Invariant Validation - Allow for the building of a snapshot of the system to test invariants.
- Dynamic Pre-/Post-condition Validation - Again allows for snapshot of system and the testing of OCL pre/post-conditions.
- Test Automation - Allows for automated Checking of system test results against the specification.
- Code Verification and Synthesis - Verify safety-critical development projects.

We add one more feature, which we deem important due to the importance of generative programming in MDE, the facility to transform UML and OCL to programming code.

Richters uses the aspects of tool support outlined above to evaluate a selection of OCL tools [Richters, 2001]. In our work, we will further this work by adding in additional tools and updating the results. The tools we will look at are:

- UML Specification Environment (USE), from the University of Bremen [USE, 2008].
- Open Source Library for OCL (OSLO) - An OCL tool which allows for the evaluation of OCL against UML2 models. This project is managed by Fraunhofer Institute FOKUS [FOKUS, Fraunhofer Institute, 2006].
- Octopus - Developed by Warmer and Kleppe, implemented as an eclipse plug-in [Warmer and Kleppe, 2006].
- EMF Validation Project - The eclipse validation project allows for EMF model constraints defined in OCL and Java [Steinburg et al., 2008].

- KeY Project - this project aims to allow for the formal verification of object software as seamlessly as possible, one of the applications of this is an OCL tool, which translates OCL to first-order predicate logic [Beckert et al., 2002].
- Dresden OCL Toolkit - designed to be integrated into other tools as an OCL library. There are several applications built on the Dresden OCL Toolkit including an OCL2 workbench, an OCL checker GUI, a Java code generator and a transformation framework. [DresdenOCL, 2007].

In table 2.2, we have outlined the aspects of tool support provided for by each of the tools described above. Question marks in the table indicate unavailable data.

Table 2.2: Analysis of OCL Tools

Feature	Tools					
	USE	OSLO	Octopus	EMF	KeY	Dresden
Syntactic analysis	yes	yes	yes	no	no	yes
Type-checking	yes	no	yes	no	no	yes
Logical consistency checking	no	no	no	no	?	no
Dynamic invariant validation	yes	no	?	yes	yes	yes
Dynamic pre/post-condition validation	yes	no	?	yes	yes	yes
Test automation	yes	no	?	no	yes	no
Code verification and synthesis	no	no	no	no	yes	no
Model transformation facility	no	no	yes	yes	no	yes

2.5 Model Driven Engineering

Model Driven Engineering (MDE), also known as Model Driven Development (MDD) and Model Driven Software Development (MDSD) is an approach to software development, where software is modelled at progressively lower levels of abstraction, eventually at the level where code can be generated from the models. In this thesis we will outline how MDE technologies and approaches provide for interoperability between TEL and the DSML defined for courseware validation. The thesis also outlines how our MDE can be used in courseware construction.

One specific MDE approach is Model Driven Architecture (MDA). MDA is an MDE approach promoted by the Object Management Group (OMG) [Frankel, 2003]. It involves using the OMG family of modelling languages to define and subsequently generate software. In order for UML to be used as a development artefact rather than a mere design artefact, UML models must be unambiguously defined. To allow for this OCL is defined on UML.

In MDA three types of models are defined in software development, Computational Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). The CIM model is a requirements model. The CIM can then be used to generate the PIM model, a systems design model, which is independent of any technology platform. The PIM is used to generate the PSM, which is the systems design model, modelled around some technology platform, such as J2EE or .NET. Model transformations are used to transform from one model type to another.

2.6 Model Transformations

Model transformations play a key role in MDE, allowing for [Czarnecki and Helson, 2006]:

- Generation of lower-level models (and code) from higher level models - as is the case in MDA's CIM, PIM and PSM.
- Definition of mappings among models at the same level or different levels of abstraction - allowing for mapping from one modelling notation to another modelling notation.
- Query based views of a system.
- Performing of model evolution tasks, such as model refactoring.
- Reverse engineering of lower level models into higher level models.

Figure 2.3, outlines the basic premises of a model transformation, where the model transformation definition is defined between two metamodels, and the transformation is then invoked at the model level.

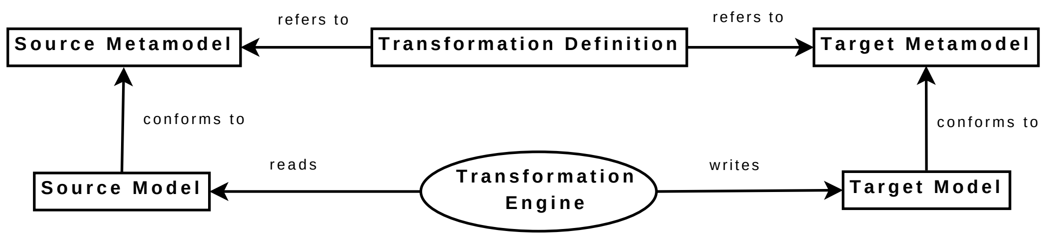


Figure 2.3: Basic concepts of model transformation [Czarnecki and Helson, 2006]

When considering a model transformation language it is important to assess the transformation language in terms of its features. To this effect Czarnecki and Helsen compare nearly thirty transformation languages using the following features [Czarnecki and Helson, 2006]:

- Specification - Is there a detailed specification mechanism for the transformation?
- Transformation Rules - Looks at how transformation rules are formulated.
- Rule Application Control - Two dimensions, local determination (the model locations that a transformation is applied to) and scheduling (scheduling the order of transformations).
- Rule Organisation - General structuring issues.
- Source-Target Relationship - Looks at how the source and target model are related.
- Incrementally - Ability to update target model with updates on the source model.
- Directionality - Are transformations unidirectional or bi-directional?
- Tracing - Mechanisms for recording different aspects of transformation execution.

2.7 Chapter Conclusion

In this chapter we have outlined the courseware terminology and concepts used in this thesis, and subsequently outlined the software engineering technologies and approaches used.

Initially, we looked at how courseware can be defined in terms of its scope, structure, content and pedagogical strategy. We also examined how courseware quality is measured. The two main courseware specifications defined to allow for the portability of courseware definitions were then described, these being SCORM and IMS LD.

The second part of the chapter detailed the software engineering technologies used in our research to allow for courseware validation. These are principally software modelling technologies used in MDE.

Software modelling languages allows for the formal and unambiguous definition of software systems and their requirements. Software modelling languages provide the ideal semantic expressivity for defining conceptual models to validate courseware. We use software modelling technologies to capture courseware design and its requirements because of the similarities between courseware and software, such as:

- Software is designed to satisfy some business need which is established though detailed analysis, while courseware is defined to satisfy some learning need established through an analysis of a skills gap.
  - Software like courseware has some specific start-point depending on its users.
  - Software like courseware has some specific end-point depending on its users.
- Time and money is saved through the reuse paradigm now prevalent in software engineering, found in software development approaches such as Component Based Software Development (CBSD) [Szyperski, 2002]. Courseware construction looks to reuse as much content and instructional design as possible to boost the quality of the courseware and to save on time and development costs [Motelet et al., 2007].
- Software models such as UML can model software structurally and behaviourally, which is the two main ways of expressing courseware.
- MDE technologies can be used to generate courseware specifications [Melia et al., 2006, Melia and Pahl, 2006b].

It should be noted that although there are similarities between the reuse paradigm in software engineering and courseware design at a component level, particularly when comparing LOs with software components such as objects, there are some major differences in their usage and properties. Sosteric & Hesemeier discuss these differences and of the incorrect comparison some authors have drawn between LOs and software objects [Sosteric and Hesemeier, 2002].

In chapter 5 we will look at the application of software models to the definition of a DSML for describing courseware and its requirements. Then in chapter 6 we outline how OCL can be used to define courseware validation criteria. In chapter 8 we will demonstrate how MDE technologies in general allow for the integration of courseware validation with the state of the art in courseware construction.

Chapter 3

State of the Art

3.1 Introduction

In this chapter we survey related research, initially looking at courseware construction (or authoring) and then at courseware validation.

Our particular focus is to examine where and how courseware validation fits into courseware construction. After looking at courseware construction, we concentrate on the courseware validation state of the art in section 3.3. We examine four approaches to courseware validation found in the literature. To analysis the state of the art examined in this chapter in section 3.4 we compare all the courseware construction and validation approaches covered in this chapter using a comparison framework. The chapter concludes by summarising our findings.

3.2 Courseware Construction

In this section we examine the state of the art in courseware construction research to provide the courseware construction context for courseware validation.

We will examine each courseware construction approach in terms of its method, the courseware granularity level it addresses and look at examples of applied research demonstrating an implementation of the courseware construction approach. Each implementation will be examined in terms of its user interface, its interoperability with other TEL tools, such as LMSs, and any validation functionality provided by the implementation. When examining an implementation's user interface we are concerned with the level of abstraction the course creator constructs courseware at. We are not interested in the tools usability, such as the tool's User Interface (UI) layout. We investigate interoperability from the point of view of how courseware validation can be integrated with courseware construction. These are central concerns to our investigation and are used to evaluate our research.

3.2.1 Specification-based Courseware Construction

In section 2.2.6, we outlined the various standardisation efforts in courseware construction. Surrounding the development of these standards has been the development of tool support for course creators to produce standard/specification compliant courseware or learning content. This tool support bases its courseware construction paradigm around a TEL specification/standard model and work at the same granularity level as the courseware specification. The primary goal of such a tool is to provide an intuitive user interface for defining the TEL specification on which it is based. The primary motivation for using the TEL specification is that courseware produced using such a tool can easily be exported in a specification-compliant way and then imported and delivered using any LMS, which is interoperable with the TEL specification used.

In the following sections we will look at the tool support developed based on the SCORM specification (see section 2.2.6.1) and then look at tool support based on the IMS Learning Design (LD) (see section 2.2.6.2) specification.

3.2.1.1 SCORM Authoring Tools

SCORM is a very popular TEL specification for courseware packaging. For this reason tool support based on the specification is numerous. Here we look at some of the more popular ones.

The Reload Project is a JISC funded project, managed by the University of Bolton [RELOAD Project, 2005]. The Reload Editor allows the course creator to create a SCORM

2004 package without knowledge of the specification's XML bindings. This is achieved through a graphical user environment that presents the SCORM XML as a file tree type structure, where the XML tree is mapped to the file tree structure. The course creator can open out elements of the XML specification, and drag and drop Learning Objects (LOs) into a SCORM package - represented as the tree structure. A comparison can be drawn to a classic file system interface, where the user can drag and drop files into various system directories. When all the LOs needed for the courseware have been added, the course creator can then define sequencing information, in the form of an IMS Simple Sequencing definition, using a form-based UI.

Figure 3.1 depicts a screen-shot of the Reload tool during courseware development. As seen in the screen-shot the course is divided into *organizations* and *items* in an *organization* as in the SCORM specification.

The Reload editor allows the course creator to create a SCORM package at a slightly higher level of abstraction, abstracting the XML implementation details. The course creator must still understand how to implement central concepts of courseware packaging according to the SCORM specification. The Reload tool does not allow for any kind of validation of the SCORM courseware created.

InSite Studio from Mississippi State University [Mississippi State University, 2007], is a diagram-based tool for the production of adaptive, modular learning content. The diagram view presents the course creator with a visual representation of the presentational flow of learning content, in flow-chart type notation. This type of presentational flow notation, allows the course creator to develop courseware in a programmatic way using conditions and actions. InSite essentially provides a simple modelling notation for defining IMS Simple Sequencing [IMS, 2003c] within SCORM. InSite does not allow for any kind of validation of the SCORM courseware created.

XML SCORM Studio [Eifel, 2007] specialises in converting legacy content into SCORM content that can interoperate with a LMS. The XML SCORM Studio also allows the course creator to create SCORM packages using a similar tree based approach to that found in the Reload Editor. There is no validation functionality in the XML SCORM studio.

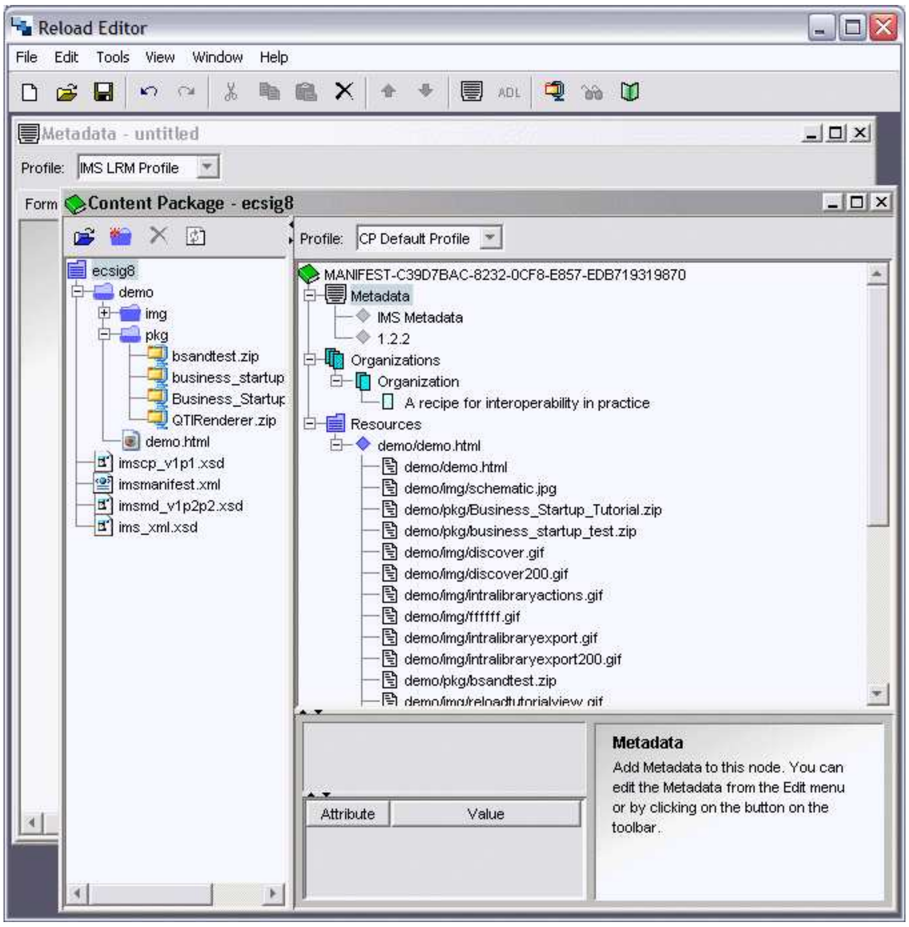


Figure 3.1: Screen-shot from the Reload Tool

3.2.1.2 IMS LD Authoring Tools

The IMS LD specification defines learning environments, including a definition of how and when to deliver learning resources to the learner. IMS LD is a more complex language than SCORM, which makes its definition more difficult, in this section we look at tool support provided for defining IMS LD.

The Reload Project, has also developed an IMS LD editor. This tool works on the same principle as the SCORM editor, in that it provides a file tree type user interface on top of the LD XML specification. This raises the level of abstraction the course creator defines LD at, from the XML level to the conceptual level. In order for the course creator to use this tool he or she must be knowledgeable on the LD specification. There is no way to validate

LD courseware created in Reload.

ReCourse is the successor to the Reload Project's IMS LD editor [Griffiths et al., 2009]. The focus of the design of ReCourse was on usability. To this end ReCourse provides a more intuitive model-based User Interface for defining IMS LD. There is also no way to validate courseware created in ReCourse.

CopperAuthor is another tool for defining IMS LD [Van der Vegt and Koper, 2005]. CopperAuthor was developed at the Open University of the Netherlands. It provides the course creator with assistance in editing an IMS LD design through a rather simplistic table interface, based on the IMS LD standard. The support provided to the course creator with this tool is minimal, providing an alternative interface to define IMS LD, than the native XML. CopperAuthor has a validation function that only allows the course creator to check that the IMS LD definition complies with the IMS LD syntax specification.

3.2.2 Adaptive Educational Hypermedia (AEH) Courseware Authoring Tools

Adaptive Hypermedia (AH) is an area of research that looks at adapting a hypermedia page to a user model, for example eliminating hyperlinks that are not relevant to a particular user [Brusilovsky, 1996]. Adaptive Educational Hypermedia (AEH), uses AH technologies in an educational context, for example, using a learner's prior knowledge to define an educationally-oriented hypermedia environment to present to the learner.

In general AEH systems operate at a low level of granularity and typically adapt to a learner's knowledge at the lesson navigation level. This is typically done by providing recommendations for a pedagogically sound learning path through the educational hypermedia to the learner. Examples of such an AEH systems are Brusilovsky et al.'s Interbook [Eklund and Brusilovsky, 1999] and ELM-ART [Weber and Brusilovsky, 2001] systems and DeBra and Calvi's AHA! system [DeBra and Calvi, 1998]. The AHA! system also works at the level of the content unit in AEH, adapting even the text presented to the learner depending on the learner model.

Much of the research in AEH, concentrates on delivery, and the effect AEH-based personalisation has on learning. For this purpose, AEH courses are generally once off implementations developed by an AEH researcher. One of the main criticisms of AEH is that its authoring is a time consuming and complex activity [Brusilovsky et al., 1998]. Here we look at one AEH authoring system, the My Online Teacher (MOT) system, that attempts to alleviate these problems in AEH authoring.

3.2.2.1 My Online Teacher (MOT)

The “My Online Teacher” (MOT) tool [Cristea et al., 2003a], developed at Eindhoven University of Technology, allows course creators to create adaptive courseware using the “LAOS” system of layered models [Cristea and de Mooij, 2003]. In LAOS there are two types of models - static and dynamic. The static models describe domain, pedagogical and learner data, and also the potential delivery environments. The dynamic model describes how AEH should adapt to variations in the static models. To create AEH using MOT, the course creator defines the static elements of LAOS using MOT. The LAOS models are layered on top of each other, with each modelling layer building on the models defined below it. LAOS’s static models are as follows, starting with the model at the bottom of the LAOS stack:

- Domain Model - Organises and structures knowledge in a particular area, defined in terms of concepts. Concepts contain attributes that contain learning content.

- Goal and Constraints Model - Used to express educational goals. This is done by specifying weights on domain concepts, optional elements in the domain model and conceptual sequencing definitions.

- User Model - Used to define the learner knowledge levels, interests and learning styles.

- Presentation Model - Model variables to do with different AEH delivery environments.

Dynamic modelling elements of LOAS models are encapsulated in the adaptation model. This model describes how the AEH reacts to variations in the static modelling elements.

The adaptation model is based on the 3-tier LAG model of adaptive specification. At the top level of the LAG model is adaptation strategies, which are collection of adaptation languages, that in turn, are collections of direct adaptation rules.

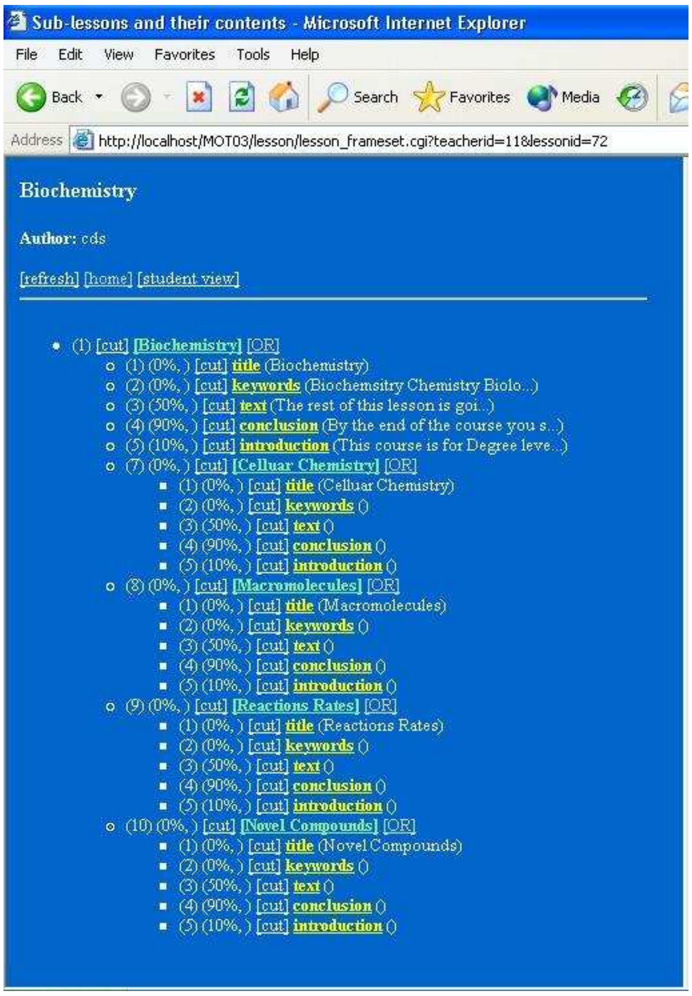
The MOT uses a hypertext interface to manipulate the LAOS models. The course creator can navigate around the LAOS models using a variety of links, where each link is a connection to another element of a LAOS model. The model is edited using a form based interface. An example of such an interface can be found in figure 3.2, where the domain model for biochemistry is being defined. The domain model has been overlaid with a goal and constraint model, indicating pedagogical information for this domain. The percentages indicate weights for concepts and attributes within concepts. Each hyper-link causes a navigational action, bringing the course creator to a new concept, attribute or an editing action to change the models in some way.

Although the MOT tool does not allow for the creation of TEL specification compliant courseware, such as SCORM packages, there has been an effort to deliver AEH courses created by MOT on different AEH delivery platforms, including AHA! [Cristea et al., 2003a] and WHURLE [Cristea et al., 2003b]. Two main approaches are taken in order to migrate the adaptive courses to the different delivery environments. The first is to use a common AEH language, the other is to use converters to convert the AEH definition in LAOS to the data structure expected by the delivery platform. MOT does not allow the course creator to check the AEH he or she has created for problems. To do this the AEH must be exported to an AEH delivery system and test-runs of the AEH must be performed by the course creator.

3.2.3 Ontology-based Authoring of Courseware

The basic idea of ontology-based courseware authoring is to use structured knowledge in the form of an ontology as a basis for creating courseware [Pahl and Melia, 2006]. The use of an ontology provides the following advantages in courseware construction:

- Domain ontologies can be used from other contexts, eliminating the courseware authoring cold start problem.



http://localhost/MOT03/lesson/lesson_frameset.cgi?teacherid=11&lessonid=72

Biochemistry

Author: cds

[refresh] [home] [student view]

```
• (1) [cut] [Biochemistry] [OR]
    ◦ (1) (0%, ) [cut] title (Biochemistry)
    ◦ (2) (0%, ) [cut] keywords (Biochemsitry Chemistry Biolo...)
    ◦ (3) (50%, ) [cut] text (The rest of this lesson is goi...)
    ◦ (4) (90%, ) [cut] conclusion (By the end of the course you s...)
    ◦ (5) (10%, ) [cut] introduction (This course is for Degree leve...)
    ◦ (7) (0%, ) [cut] [Celluar Chemistry] [OR]
        ▪ (1) (0%, ) [cut] title (Celluar Chemistry)
        ▪ (2) (0%, ) [cut] keywords ()
        ▪ (3) (50%, ) [cut] text ()
        ▪ (4) (90%, ) [cut] conclusion ()
        ▪ (5) (10%, ) [cut] introduction ()
    ◦ (8) (0%, ) [cut] [Macromolecules] [OR]
        ▪ (1) (0%, ) [cut] title (Macromolecules)
        ▪ (2) (0%, ) [cut] keywords ()
        ▪ (3) (50%, ) [cut] text ()
        ▪ (4) (90%, ) [cut] conclusion ()
        ▪ (5) (10%, ) [cut] introduction ()
    ◦ (9) (0%, ) [cut] [Reactions Rates] [OR]
        ▪ (1) (0%, ) [cut] title (Reactions Rates)
        ▪ (2) (0%, ) [cut] keywords ()
        ▪ (3) (50%, ) [cut] text ()
        ▪ (4) (90%, ) [cut] conclusion ()
        ▪ (5) (10%, ) [cut] introduction ()
    ◦ (10) (0%, ) [cut] [Novel Compounds] [OR]
        ▪ (1) (0%, ) [cut] title (Novel Compounds)
        ▪ (2) (0%, ) [cut] keywords ()
        ▪ (3) (50%, ) [cut] text ()
        ▪ (4) (90%, ) [cut] conclusion ()
        ▪ (5) (10%, ) [cut] introduction ()
```

Figure 3.2: Screen-shot of MOT tool showing a lesson map with each attribute given a weight

- The semantics within the domain ontology can be used to find learning content.
- The domain ontology can be used as a principle navigation tool in the courseware created.

The tools outlined in this section demonstrate how ontologies are used in the construction of courseware and fine-grained learning resources such as lessons or LOs.

3.2.3.1 TANGRAM

TANGRAM is a tool that has been developed by Jovanović et al. to aid course creators in reusing and repurposing content units in LOs [Jovanović et al., 2006b]. TANGRAM attempts to cater for the course creator who wishes to create a new LO from content units of other LOs by decomposing LOs into small, highly reusable, content units. These content units are then automatically annotated and can be searched by the course creator when creating a new LO.

Decomposition of LOs is based on the ALOCoM Content Structure ontology definition [Jovanović et al., 2005]. Once LOs have been decomposed into content units TANGRAM is able to automatically annotate each content unit. Annotation is based on the IEEE Learning Object Metadata (LOM) standard [IEEE LTSC, 2002]. A domain ontology is used to capture subject domain information, and a context ontology is used to capture the educational context of the content unit, such as whether a content unit is a paragraph or a title of presentation slide. The automatic generation of LOs is done using ontology-based inference techniques.

The TANGRAM tool produces LOs in the form of openoffice[1] documents. The course creator tests the LOs produced by TANGRAM by exploring them as a learner, there is no automated validation functionality in TANGRAM.

3.2.3.2 OntAWare

In [Holohan, 2003], Holohan describes how structured knowledge such as ontologies can be used to create courseware using the OntAWare system. Holohan has developed an algorithm that takes, as input, a tree-like ontological structure, and generates Microsoft PowerPoint slides that reflects the knowledge in the ontology. Holohan et al. also demonstrates how ontologies could be used to create multiple-choice questions to test student's knowledge on a given subject [Holohan et al., 2006, Holohan et al., 2005].

More recently an ontology-based delivery system has been developed for OntAWare.

[1]`http://www.openoffice.org`

The OntAWare delivery system creates an “active slide” of information about an ontology component on the fly as the learner navigates the ontology. This means changes to the knowledge base will be reflected in any of the learning content delivered by OntAWare [Melia et al., 2005]. The delivery of the courseware can also be personalised for the learner in terms of the learner’s knowledge [McMullen, 2007].

The tool provides the course creator with a form-based user interface for editing the course knowledge base and for editing the courseware instructional design. The editing interface is quite intuitive, but does have scalability issues as the course creator has no way of viewing the knowledge base at a higher level of abstraction other than an individual concept level. Courseware constructed using OntAWare can be exported into SCORM 1.2 packages, to be delivered on LMSs compliant with that specification [McMullen et al., 2005].

To test the courseware produced by OntAWare the course creator must perform dummy learner runs to see how the courseware adapts to learner attributes. There is no way to automatically validate the courseware using OntAWare.

3.2.3.3 Visualized Online Authoring Toolkit (VOAT)

Yang et al. recognise the difficulties in reusing and repurposing LOs in [Yang et al., 2005], and have created a suite of tools which aim to aid the course creator in the creation of courseware. Yang and his team have embraced the Semantic Web movement in their suite of tools by using Semantic Web ontologies as the basis for courseware creation. The tool suite has been built on top of one of Yang’s earlier attempts to encourage LO reuse and repurposing, the SCORM-compliant Content Repository Management System (CRMS) [Yang and Tsai, 2003].

There are three components to VOAT; the Ontology-Based Outline Authoring Tool (OBOAT), the Visual Online Course Authoring Tool (VOCAT) and the Visualized Online Simple Sequencing Authoring Tool (VOSSAT). The course creator uses OBOAT to create a conceptual map of the content of the course to be created. The construction of this conceptual map is guided by a domain ontology. The structure of the course the ontology is displayed in a file tree type structure, such as that found in the Reload tool.

Once a conceptual map has been created it is loaded into the VOCAT, which is used to add and assemble LOs into courseware. Editing of the courseware in VOCAT is done through a set of eight buttons that are based around the addition and subtraction of LOs to the course. There is a steep learning curve in using VOCAT as the course creator must learn about the functionality of each button offered and their symbols.

The final tool in the suite is VOSSAT, which allows the course creator to specify LO sequencing information. Sequencing information is specified through a form-based user interface. Once sequenced the courseware is exported into SCORM and can then be used by learners on a SCORM-compliant LMS. VOSSAT has a form-based interface that uses the terminology of the IMS simple sequencing specification. To use VOSSAT, in-depth knowledge of the simple sequencing specification is required.

VOAT concentrates on SCORM-compliant courseware construction. It is not concerned with checking the courseware constructed for problems. As such, there is no validation functionality in VOAT.

3.2.4 Constraint-based ITS

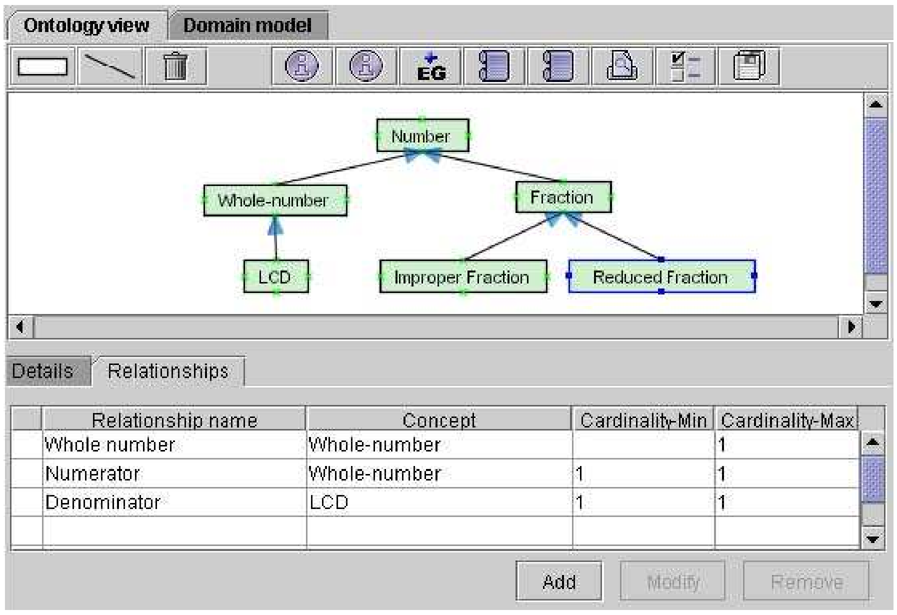
The SQL-Tutor, developed at the University of Canterbury, is an Intelligent Tutoring System (ITS) for learning to program SQL for relational database systems [Mitrovic et al., 2007]. This ITS is defined through constraints. Constraints work by checking the existence of problems in the solution provided by the learner, violated constraints give clues to how to support the learner. The constraint based approach of only correcting the learner for known problems means if the learner does something that the system doesn't know about it, it deems it to be correct - the learner is therefore “innocent until proven guilty” [Mitrovic et al., 2007]. In order for the course creator to define an ITS for learning SQL almost 700 constraints have to be defined, describing the fundamental principles of SQL.

Constraint-based Intelligent Tutoring System (ITS) authoring support is provided by the WETAS-ontology [Martin et al., 2007] and ASPIRE [Mitrovic et al., 2006] systems. The WETAS-ontology allows the course creator to define an ontology for the desired ITS domain and generate constraints from the ontology definition [Martin et al., 2007]. ASPIRE is a complete authoring system for a constraint-based ITS. ASPIRE looks to be as intuitive as possible, allowing the course creator to define their own ITS in nine steps, as follows:

1. Specify the domain characteristics - The central domain concepts are broken down into tasks, steps or parts.
2. Compose a domain ontology - An ontology is defined to represent the subject domain.
3. Model the problem and solution structures - Define how domain problems and their corresponding solutions and structures are declared.
4. Design the student interface - Based on the problem and solution structures the student UI is designed.
5. Add problems and solutions - Example problems and solutions are added.
6. Generate the syntax constraints - Syntax constraints are generated based on the domain ontology definition.
7. Generate semantic constraints - Machine learning is used to derive semantic constraints from the problems and solutions provided by the course creator.
8. Validate the generated constraints -The system is tested by the course creator to ensure it can identify an incorrect solution.
9. Deploy the tutoring system - ITS is deployed in the ASPIRE-Tutor system.

The ASPIRE UI is mainly form-based, where much of the steps above have a form associated with them in ASPIRE. The domain ontology editing part of ASPIRE provides a graphical model of the ontology as shown in figure 3.3. The syntax and semantic constraints are displayed in their native LISP [Steele, 1990] and can be edited by the course creator.

The constraint-based approach allows the learner to be right, unless the system “knows” the learner is wrong. This means that every wrong approach must be explicitly defined for the course to notify the learner when he or she does something wrong. The approach is excellent for smaller courses, but as seen with the SQL-tutor system, the approach does



Relationship name	Concept	Cardinality-Min	Cardinality-Max
Whole number	Whole-number		1
Numerator	Whole-number	1	1
Denominator	LCD	1	1

Figure 3.3: Domain model editing in ASPIRE

have scale problems. The constraints defined in ASPIRE are solely used in defining the ITS, the constraints cannot be used for validating the ITS in terms of its pedagogy. Validation of the ITS can only be achieved through manual testing.

3.2.5 Model Driven Courseware Engineering

Courseware engineering is the use of software engineering technologies and methodologies in the development and maintenance of courseware [Dwolatzky et al., 2002]. One of the main software engineering principles used in courseware engineering is abstraction, where abstraction is a deliberate simplification that picks out the most salient characteristics of a system [Sommerville, 2004, p170]. Abstraction is used to remove excess complexity in designing courseware.

Courseware engineering generally works at the courseware or module granularity level, which mainly focuses on the combination of learning resources into a focused unit of instruction.

Here, we look at one particular type of courseware engineering, the use of Model Driven Engineering (MDE) technologies to capture courseware structure and the learner flow through the courseware, and the subsequent generation of a courseware specification based on the model. Melia et al. and Martínez-Ortiz et al. have looked at the use of UML Activity Diagrams to define IMS Simple Sequencing (SS), and investigate how software model transformation technology can be used to generated a IMS SS XML binding [Melia et al., 2006, Martínez-Ortiz et al., 2009]. Further insights into the use of MDE can be found in [Laforcade and Choquet, 2006], which looks at the use Model Driven Architecture (MDA) methodologies in courseware development.

Here we will examine approaches to courseware creation that are based around the definition of data models that capture the courseware construction concerns. These data models are then used to create courseware.

3.2.5.1 Adaptive Courseware Construction Toolkit (ACCT)

The design of the Adaptive Courseware Construction Toolkit (ACCT) is based on the separation of the "key design elements of personalised elearning" [Dagger, 2006b]. These elements are (but not limited to) as follows:

- Narrative Structures - Pedagogical structures that specify a sequence of learning activities and tasks.

- Activities - Activities are usually some task that must be performed in the e-learning environment. There can be tools associated with particular tasks.

- Subject Matter Concept Space (SMCS) - Knowledge about the subject area. The (SMCS) also provides scope for the the adaptive courseware and limited pedagogic information such as dependencies between concepts.

- Narrative Attributes - Narrative attributes describe some form of adaptation. It describes information needed to make decisions on adaptivity, which is passed to "selectors" that select the most appropriate content from candidate content groupings [Dagger et al., 2003].

- Learning Resources - Learning resource metadata can be used as the basis for adaptation.
- Learner Profiles - A learner profile describes a learner in terms of the knowledge the learner has.
- Personalised e-Learning Designs (PEDE Narrative) - The resulting personalised course based on the above personalised e-learning design elements. The PEDE narrative captures all the information needed to deliver the adaptive course. The PEDE narrative typically operates at the module level of courseware.

Figure 3.4 shows a screen-shot of the ACCT, and highlights the use of each of the personalised e-learning design elements.

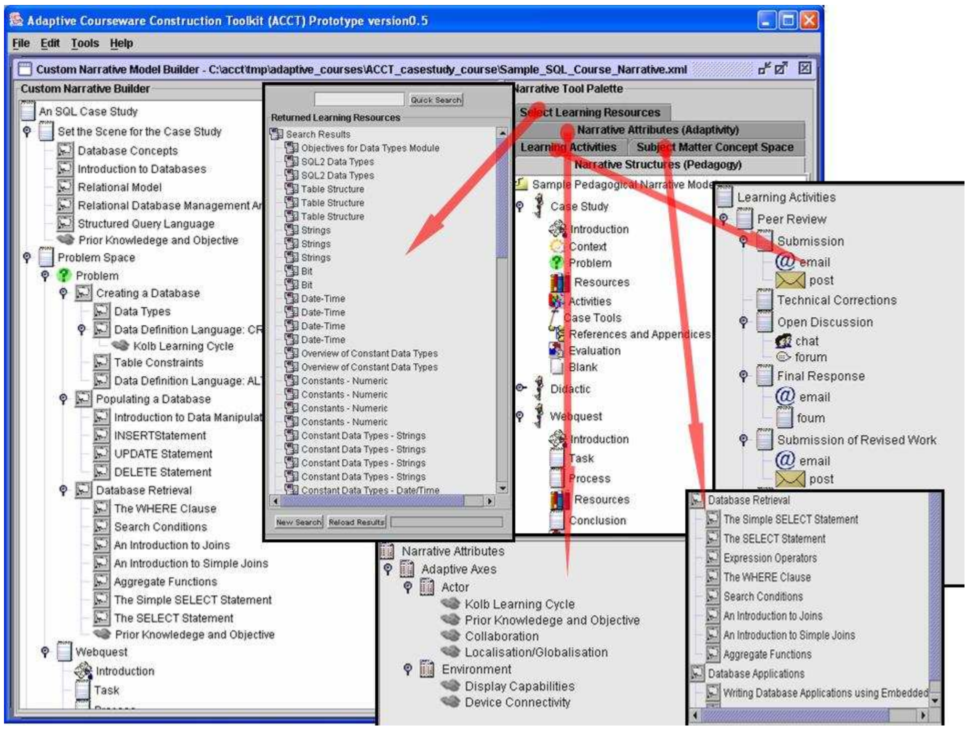


Figure 3.4: The ACCT - Demonstrating the use of PEDE design elements [Dagger, 2006b]

Each design element in the ACCT is not dependent on any other design element, which makes design elements very portable. For example, the narrative structure that specifies a good web-based learning pedagogy (Web Quest) can be used in other non-related courses.

The combination of a concept map with pedagogic information, found in the ACCT, can limit the portability of a SMCS to only those courses where the course creator agrees with the set pedagogy.

To test courseware constructed using the ACCT, it is deployed into the "gAPeLS" delivery system allowing the course creator to perform dummy runs of the new courseware.

The ACCT is interoperable with mainstream TEL, as personalised courseware defined using the ACCT can be exported to IMS LD [Dagger, 2006a]. Although there are some limitations in the solution, the integration of the ACCT with IMS LD shows the importance of interoperability with courseware specifications for courseware construction tools.

3.2.5.2 MD2 Course Authoring

The MD2 method allows the course creator to create didactic materials using a model-based approach [Padrón et al., 2006]. The method aims to support the course creator in a four-phase process. The first phase is an analysis of the subject area, the second phase involves selecting the resources to be used according to pedagogical and technical requirements. In the third phase learning materials are composed into a course and finally in the fourth phase the didactic material created is evaluated.

The MD2 method uses the MD2 model descriptors to describe the requirements for the didactic material. The MD2 model has four main types of descriptors:

- Knowledge Domain Descriptors - Represents the main features of the discipline or subject to be taught. This descriptor is used as a reference knowledge structure.

- Pedagogic Descriptors - Captures the pedagogical requirement of the didactic material being created, this descriptor is akin to the educational category of the IEEE LOM standard [IEEE LTSC, 2002].

- Support Descriptors - Ensures the didactic material created is reusable, interoperable and manageable.

- Usability and Quality Descriptors - Describes the constructed didactic material in terms of its usability and also check if the created material can effectively support educational tasks in order for the material to achieve its objective.

The MD2 method implements the four-phase process in thirteen steps, with various steps associated with each phase. These steps are centred around the definition of a MD2 model. Tool support is provided for the MD2 method. This tool uses a questionnaire interface to quiz the course creator on the didactic material requirements, building up a MD2 model that reflect the didactic material requirements. This is a very simple interface for the course creator, but can be very restrictive, as the course creator can not add any information to the MD2 model that is not covered in the questionnaire.

The final step of the MD2 process is the only step concerned with the evaluation phase. This step defines the usability and quality descriptors of the MD2 model. This step is based on learner experience. To evaluate the didactic material's quality the course creator must evaluate if the material defined "effectively supports educational tasks and helps to achieve the defined learning objectives" [Padrón et al., 2007]. The usability elements provide the course creator with the facility to evaluate the didactic material created in terms of; error frequency and severity, self-evidence, efficiency of use, familiarisation time and memorability. All values are represented as fuzzy sets and are aggregated to obtain a usability and quality value for the material. This is a form of validation of the material that is created at the post-delivery stage.

The importance of evaluation in MD2 is demonstrated in [Padrón et al., 2008], where the usability and quality descriptors are used to drive adaptation, replacing below par material during the delivery of courseware.

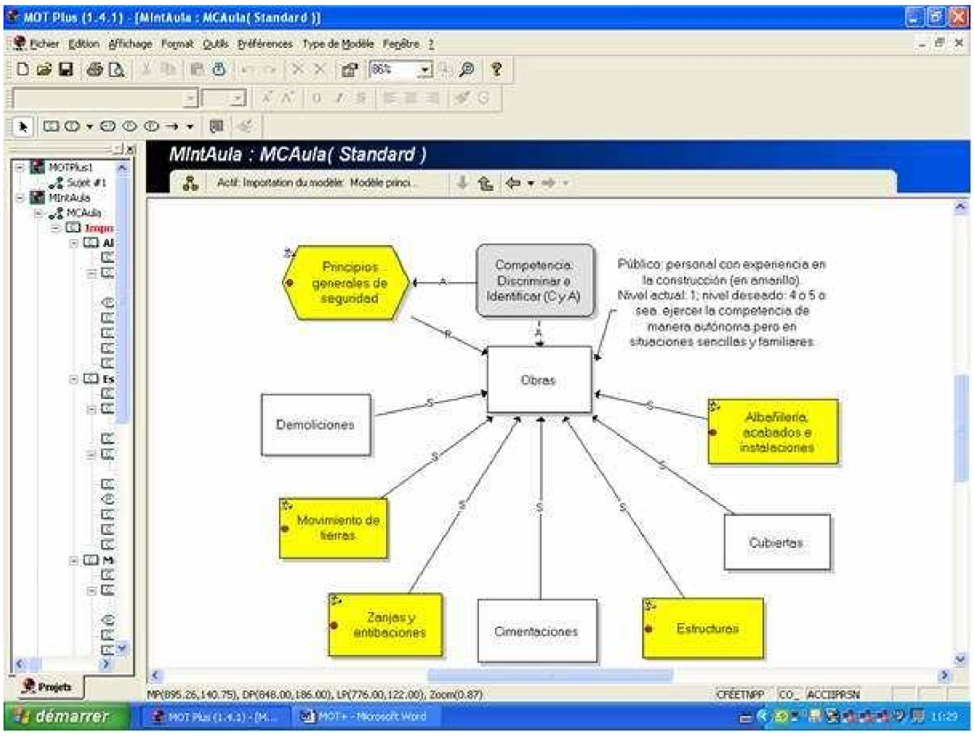
Padrón et al. mention the use of IMS LD XML bindings as a method to represent delivery and presentation patterns. Should this be implemented it would allow for interoperability with any IMS LD delivery platform [Padrón et al., 2006].

3.2.5.3 MOT+

MOT+ is a graphical knowledge modelling language for instructional engineering [Paquette et al., 2006]. It is worth noting that MOT+ has no relation to the My Online

Teacher (MOT) tool described in section 3.2.2.1. MOT+ can be applied in a multi-actor process model aiming to improve the learning design produced. The MOT+ can represent concept, procedure and principle knowledge types, which are related to each other through various relationships such as, specialisation link, precedence link, and input-product link. These basic model constructs are used to build various knowledge models, such as factual models, procedural models and processes.

The MOT+LD editor, uses MOT+ defined models, processes and methods to define IMS LD. A graphical modelling language is defined for LD. The course creator can use this vocabulary to define the various LD constructs he or she needs in order to fulfil the learning objectives defined for the courseware being constructed.


MOT Plus (1.4.1) - [MIntAula : MCAula(Standard)]
Fichier Edition Affichage Format Outils Préférences Type de Modèle Fenêtre ?
MIntAula : MCAula(Standard)
Actif: Importation du modèle: Modèle princi...
Principios generales de seguridad
Competencia: Discriminar e Identificar (C y A)
Obras
Demoliciones
Albañilería, acabados e instalaciones
Movimiento de tierras
Cubiertas
Zanjas y entibaciones
Cimentaciones
Estructuras
A
R
S
MOTPlus1
Sujet #1
MIntAula
MCAula
Projets
démarrer
MOT Plus (1.4.1) - [M...
MOT+ - Microsoft Word

Público: personal con experiencia en la construcción (en amarillo)
Nivel actual: 1; nivel deseado: 4 o 5 o sea, ejercer la competencia de manera autónoma pero en situaciones sencillas y familiares

MP(895.26,140.75), DP(848.00,186.00), LP(776.00,122.00), Zoom(0.87)

Figure 3.5: Screen-shot of the MOT+ tool

The MOT+ graphical representation provides a usability benchmark for defining a courseware construction modelling language, as MOT+ “has been proven sufficiently simple and friendly to be used by persons with non-technical background” [Paquette et al., 2006], such as the majority of course creators. A screen-shot of the MOT+ tool can be found in figure

3.5.

Tool support for MOT+ does not provide the course creator with any form of courseware validation functionality.

3.2.5.4 Sequencing Objects

Su et al. look at how abstraction can be applied to defining courseware sequencing strategies in [Su et al., 2005]. This research recognised the steep learning curve faced by the course creator creating SCORM 2004 compliant courses, and suggest using the Object Oriented Methodology (OOM), in [Su et al., 2005], to simplify the creation of SCORM compliant courseware. Using the OOM, complicated sequencing rules are encapsulated into objects, known as a sequencing object. The types of sequencing objects that are available to the course creator are:

- Linear - The learner flows a set path through learning content. The learner cannot revisit learning content once completed.
- Choice - The learner chooses what to learn.
- Conditional - The course creator can set conditions which are evaluated at runtime and cause the lesson to behave in a certain way.
- Loop - Allows the course creator to force the learner to repeat aspects of a course if certain conditions are (not) satisfied.
- Exit - Allows the course creator to set conditions whereby a section of the course is complete.

The course creator must add learning content to sequencing objects, and then joins the sequencing objects together to create a courseware. This course is then translated into a SCORM 2004 package using the “Course Sequencing to Content Package” (CS2CP) algorithm.

The use of sequencing objects is an excellent example of how models can be used to alleviate some of the complexity involved in creating SCORM sequencing. Sequencing objects use a self-defined notation, and are not standardised, and therefore do not reap the benefits of the automated transformation processes developed for standardised OO modelling languages, such as the UML.

A proof of concept implementation is described briefly in [Su et al., 2005], which allows the course creator to define the course in a graph-based, flow-type graphical notation. The sequencing objects are used as patterns, which can be applied to the courseware graph representation. The user interface is intuitive, in that the course creator just needs to express the flow of a learner through courseware LOs. It is similar to the user interface used in MOT+, described in section 3.2.5.3. The implementation does not provide the course creator with any way to check for problems in the courseware constructed using sequencing objects.

3.3 Courseware Validation

In this section, we survey the state of the art research addressing validation of constructed courseware. We will outline the validation approach, looking at the courseware granularity level addressed by the approach and examples of applied research in terms of its user interface, flexibility and interoperability with other TEL tools.

3.3.1 Concept-based Course Analysis (CoCoA) Tool

The Concept-based Courseware Analysis tool (CoCoA) was developed at Carnegie Technology Education [Brusilovsky, 2000, Brusilovsky and Vassileva, 2003]. It validates a given courseware by analysing LOs in the courseware in terms of their type and how they are indexed against a domain model.

“Advanced concept roles” define how a LO is indexed using domain model concepts. A LO is indexed with regard to the prerequisite knowledge required for the LO and the knowledge outcome for a learner who engages with the LO. There are two types of pre-requisites defined and two types of knowledge outcomes defined, strong and weak. A strong prerequisite or outcome indicates that deep knowledge of the concept referenced is required, or obtained from the specified LO, while weak pre-requisite or outcome indicates that only surface knowledge of the concept referenced is required or obtained for the specified LO. There are also various levels of prerequisite constraints defined for LOs.

The LO types are recorded as “typed items”. The types of LOs are presentations, examples, assignments and multiple-choice questions.

CoCoA checks sequential learning paths through LOs in a given courseware. This is done by simulating a learner’s progression through courseware LOs. The tool can check for the following characteristics in courseware:

- Content Holes - Where a learner encounters a LO in courseware without having the necessary prerequisite knowledge needed for the LO.
- Consolidation of Presentation - Ensures concepts are just introduced once, and once a concept has been taught in detail the introductory material is not shown again,
- Question Placement and Repositioning - The system can place questions for a concept in the right place for a given courseware.
- Guidelines for Question Design - Enables the course creator to validate that the questions used in courseware cover an adequate amount of concepts.
- Matching Presentations with Examples and Exercises - Checks that examples and exercises are placed correctly in the courseware.
- Checking the Course Design Goal Against the Courseware - Allows the course creator to check that the courseware achieves its learning goals.
- Presentation Density and Sectioning - The number of concepts taught in each courseware element is not excessive and there is some consistency about the number of concepts taught at each courseware element.

CoCoA was prototypical in nature and as such there is no consideration for TEL specifications or standards. Pedagogical problems are defined by the CoCoA software developer, there is no facility for the course creator to define validation rules. This caused problems with user acceptance, as users did not agree with some of the courseware “problems” flagged by CoCoA. CoCoA was not developed using an extensible architecture, which would allow for the inclusion of unforeseen pedagogical rules in the future. It also does not reflect the complexity of validating modern courseware as all courses validated using CoCoA must be linear in nature with no branching points. However, CoCoA demonstrates the viability of courseware validation and also shows that it is not a trivial problem.

3.3.2 Logic-based Course Planning and Verification

The use of logics and reasoning for course planning and validation has been investigated by the ALICE project at the University of Torino [Baldoni et al., 2004a]. The project looks at a range of course construction activities including course validation [Baldoni et al., 2004b] and construction [Baldoni et al., 2004a] using logical reasoning.

Reasoning is performed on an ontological knowledge base. The learner and learning resources can be defined in terms of this knowledge base. Learning resources are viewed as logical actions, with pre-requisites and effects.

The Wlog system, developed by the ALICE project, demonstrates how logics can be used in courseware creation and validation. The motivation behind the Wlog system is to validate Italian students study plans for a year with respect to their overall degree course goals. A study plan is a list of courses a student takes in University. Each year students may alter their study plan, but these alteration can have adverse affects for the students overall degree learning goal. Study plan creation and validation is at a coarse granularity level in course creation and management. The Wlog system uses the DyLOG logic language [Baldoni et al., 2004c] to represent knowledge and to program its behaviour. The ALICE project, which developed the Wlog system specifies where possible conditional curricula plans are generated from an initial state to a goal state [Baldoni et al., 2004a]. The initial state outlines the student’s knowledge at course design-time. The goal state defines either the knowledge the learner wishes to acquire or the courses the learner wishes to complete. Plans are created to bring the learner from the initial state to the goal state. When there are conditional branches in the plan, the Wlog system uses sensory actions to ask the learner what branch they wish to take.

Temporal projection is used to check that all pre-conditions in the action theory are respected. Courses are represented using action theory, where each course component is an action with pre-conditions and post-conditions. Curricula models allow for restrictions and constraints to be placed on possible learning resource sequences at the knowledge level. Curricula models can be formalised using temporal constraints. The models are independent from the learning resources and operate on the knowledge level. For example a possible constraint might be that $knowledgeElement_a$ must be learned before $knowledgeElement_b$ can be attempted. Using linear-time temporal logic (LTL) to represent temporal constraints allows for the validation of the courseware in terms of the curricula models. Should the course be deemed invalid, it is important to be able to outline to the course author the reasons for invalidating the course. Temporal explanation is used to explain the reasons for a course failing validation.

An attempt has been made by Baldoni et al. to examine how this technology could be applied at lower level of courseware creation granularity [Baldoni et al., 2004b], even embracing courseware specifications and standards, such as SCORM [ADL, 2004] and the IEEE LOM [IEEE LTSC, 2002]. The main motivation for this is to see if different reasoning techniques can be used to allow for adaptation to learner stereotypes. LOs can be composed on the fly based on pre- and post-conditions of SCORM LOs using additional learning strategy constraints. The learning strategy allows the course creator to specify such things as:

- Strategies for Learner Stereotypes - for example what LOs to select for biology students learning computer programming.
- Learning Goals.
- LO Sequencing Strategies.

Although the ALICE group has looked at more fine-grained LO composition and validation, the majority of their work concentrates on the validation and construction of study plans that define a qualification course [Baldoni et al., 2006, Baldoni et al., 2004a]. This is particularly interesting in the context of the Bologna Process [EU Bologna Agreement, 2000], which requires course content to be verified against external specification, being promoted at the European level.

3.3.3 Ontology-based Guideline Definition for Courseware

Sicilia makes the observation that the rationale used to define an IMS LD specification is not captured. In [Sicilia, 2006] Sicilia outlines an ontology-based approach to model the learning design and the rationale used to create the learning design in a common ontology. The motivation for this is to allow for the generation of empirical data regarding the application of a learning theory through a learning design.

Sicilia firstly outlines how the IMS LD specification can be defined using an ontology. This ontology is defined in terms of the opencyc ontology, an open source knowledge base containing over 100,000 atomic terms [Lenat, 1996]. The opencyc ontology is extended to allow for the definition of the rationale that is used in the design of a LD specification.

Two methods for defining constraints on IMS LD definitions are provided in [Sicilia, 2007], these are models and guidelines. Models provide a template-like structure for defining a LD definition, where the LD should reflect the template when certain conditions hold. Guidelines can be defined in the context of the rationale and an LD ontology, where guidelines "describe how learning resource design and their outcomes 'should be' in a propositional form" [Sicilia, 2006]. Guidelines can be formalised by defining constraints on the IMS LD ontology, guidelines can be realised using the Semantic Web Rule Language (SWRL) syntax [Horrocks et al., 2004]. This approach allows for the definition of LD guidelines based on an instructional design theory.

Sicilia's work outlines an initial effort in defining courseware and its rationale using a common point of context. The work also demonstrates how constraint structures, in the form of "models" and "guidelines", can be used to guide the courseware design process or generate tentative courseware designs automatically [Sicilia, 2007].

3.3.4 Trap Detection in IMS Simple Sequencing

IMS Simple Sequencing [IMS, 2003c] allows for the definition of courseware sequencing behaviour. It is most commonly used in the SCORM specification. In section 2.2.6 we outlined how the IMS Simple Sequencing specification is defined. Defining adaptive courseware using IMS Simple Sequencing is not a trivial task. Course creators can unknowingly embed the following problems in a simple sequencing definition, known as sequencing traps [Lin and Shih, 2009]:

- Learning Attempt Stuck in an Activity - Even though the learning activity is complete the learner is blocked in the activity and cannot move on.
- Learning Attempt has Fallen into a Vicious Circle - A deadlock situation where a learning path has resulted in a loop that it is not possible for the learner to get out of.
- Deserved Activities are not Identified for Delivery - Sequencing rules cannot be fired and learning resources that should be delivered to the learner are not.

Lin & Shih outline a solution for the detection of these sequencing traps. This solution is based on the generation of a petri net representation for an IMS Simple Sequencing definition and the subsequent tracing of the petri net graph in search of sequencing traps. In their approach Lin & Shih separate "foundation constructs" and "operation constructs", where foundation constructs are the definition of the sequencing control mode boolean values that define the behaviour of a sequencing cluster, while operation constructs define the sequencing rules in a condition-action format. Various combination of foundation constructs and operation constructs can cause sequencing traps. Lin & Shih have identified some of these, which are outlined in [Lin and Shih, 2009]. The paper also illustrates each sequencing trap graphically using a petri net diagram.

Sequencing traps are identified in two main ways. The simplest way is when a petri net cannot be built for a given IMS Simple Sequencing specification. If a petri net representation can be built, the second way is to use the petri net in a sequencing trap detection algorithm. Lin & Shih outline a detection approach for locating sequencing traps consisting of two algorithms.

The work of Lin & Shih specialises in problems with the IMS Simple Sequencing specification, where the combination of particular attribute values in the specification causes erroneous behaviour. The approach taken by Lin & Shih is to validate the sequencing specification for these problems. The problems highlighted by Lin & Shih are a symptom of the lack of good tool support for defining IMS Simple Sequencing. Good tool support would protect the course creator from themselves when defining Simple Sequencing by raising the level of abstraction in such a way that these sort of problems could not be defined.

3.4 Comparison Framework

In this section, we present a comparison of the courseware construction and courseware validation approaches. After presenting each comparison we discuss the comparison and bring points of interest to the reader's attention.

3.4.1 Courseware Construction Approaches

Table 3.1 provides a matrix of the features for the state of the art in courseware construction covered in this chapter. The features used to compare the state of the art represent the major differences in the approaches presented and provide a useful mechanism with which to compare each approach. From this comparison we can generalise as to the most common state of the art courseware construction environment. Our courseware validation approach must be aligned to this. The comparison attributes used are as follows:

- Granularity Level - Refers to the courseware granularity level the courseware construction approach operates at.

- Internal Data Model - Many courseware construction approaches base their approach on some well defined data model. Approaches are compared based on this.

- Interoperability - Examines if the courseware construction implementation can interoperate with complementary systems or specifications.

- User Interface - Looks in general at the approach taken to define a user interface.
- LOR Support - Examines if the implementation of the approach allows for interoperability with LORs.
- External Domain Model Support - Courseware construction can be based around a domain model allowing for the automation of some of the more trivial tasks in courseware construction. This attribute looks at whether the approach allows for the use of an external domain model.
- Validation Support - Support provides for the validation of the constructed courseware.

A question mark indicates data is unavailable.

Table 3.1: Courseware construction approaches comparison

Courseware Construction							
	Granularity level	Internal data model	Interoperability	User Interface	LOR Support	Domain Model Support	Validation Support
Reload SCORM Tool	Course-lesson	SCORM specifications	SCORM 2004	Tree-based	No	No	No
Insite Studio	Course	?	SCORM 2004	Model-based (course flow)	No	No	No
XML SCORM Studio	Topic		SCORM	?	?	No	No
Reload LD Tool	Course - lesson	IMS LD specification	IMS LD	Tree-based	No	No	No
ReCourse	Course - lesson	IMS LD specification	IMS LD	Model-based	No	No	No
CopperAuthor	Course-lesson	IMS LD specification	IMS LD	Form and tree-based	No	No	LD compliance
MOT	Lesson	LAOS model	CAF	Web-based forms	No	Yes	No
TANGRAM	LO	various ontologies	open office	Web-based forms	No	Yes	No
OntAWare	LO - lesson	OWL ontology	SCORM 1.2	Web-based forms	Limited	Yes	No
ACCT	Course	variety of unconnected models	IMS LD	Model-based	Limited	Yes	No
VOAT	Lesson - course	ontology-based	SCORM 2004	Tree-Structure based	No	Yes	No
Constraint-based ITS Authoring	Lesson - course	constraints defined on a domain model	SQL Tutor	Model-based and text-based	No	Yes	No
Sequencing Objects	Lesson - course	SCORM 2004	SCORM 2004	Model-based	No	No	No
MD2	Lesson - course	MD2 Model	?	Questionnaire-based	No	Yes	No
MOT+	Lesson - course	?	IMS LD	Model-based	No	Yes	No

As outlined in table 3.1, the majority of the courseware construction state of the art operates at the “lesson” to “course” granularity level. The internal data models can be categorised into those tools, which use a TEL specification, those which are ontology-based and those using a custom-built model. The custom-built approaches highlight the courseware definition requirements currently not in the TEL specifications, such as that needed for extensive personalisation. This is also demonstrated with respect to interoperability, as some personalised and adaptive courseware features can only be delivered through specialised delivery environments, such as AEH.

Courseware construction tool support provides user interfaces that are text-based, tree-based and model-based. The level of abstraction the course creator operates at increases as you move from text-based UI up to the model-based UI, although some simplistic approaches to text-based courseware construction have been investigated in the state of the art (e.g. questionnaire). We believe the most intuitive of the user interfaces is the flow-based models found in MOT+ and Insite Studio. This approach does have limitations in that it does not scale to highly adaptive courseware, as displaying all the possible combinations of adaptivity can get quite verbose.

LOR support is somewhat lacking in the current state of the art, but this may be because of a lack of standardisation in how to interoperate with LORs we look at recent advances in this area later in this thesis (section 8.10). The use of a domain model, seems to be common practice in courseware construction tools, particularly those in the research arena. Domain models allow the course creator to quickly define what should be in the courseware and how the courseware should be structured. This allows for the rapid development of courseware.

We also note that none of the courseware construction tools surveyed allows for the validation of any kind of pedagogical features of newly constructed courseware. Only the CopperAuthor tool allows for some kind of validation, but this is purely to validate the realisation of the IMS LD specification.

3.4.2 Courseware Validation Approaches

Table 3.2 provides a matrix of the features of each of the state of the art courseware validation approaches, covered in this chapter. The comparison attributes used here are as follows:

- Granularity Level - The granularity level of the courseware that can be validated.
- Internal Data Model - How the validation approach represents the validation data.

- Interoperability - Examines if the validation implementation can interoperate with complementary systems and/or specifications.
- User Interface - Looks in general at the approach taken to define a user interface.
- Validation Criteria - Investigates if the validation criteria can be defined separate to the validation implementation framework definition.
- Domain Model Support - Investigates whether an external domain model can be used in defining courseware validation data.

Table 3.2: Courseware validation approaches state of the art comparison

Courseware Validation						
	Granularity Level	Internal Data Model	Interoperability	User Interface	Editable Validation Criteria	Domain Model Support
Logic-based Validation	Qualification course	Logical model - learning resources are actions	SCORM	Text	No	No
CoCoA	Lesson-module	Domain model based	No	?	No	No
Ontology-based guidance	Lesson-module	IMS LD	IMS LD	No	Yes	Opencyc only
Trap detection	Lesson-module	IMS Simple Sequencing	IMS SS	No	No	No

The approaches to courseware validation addressed in this chapter are prototypical in nature. Only the logic-based approach and CoCoA actually go as far as providing for a proof of concept implementation. We have also found that Interoperability is a concern that has not been addressed satisfactorily in the state of the art. The tools we have looked at either do not provide for interoperability at all or provide for only limited interoperability with courseware construction tools, considering just one interoperability specification in TEL. TEL specifications and standards are in flux, courseware validation tools must be modifiable to new specifications and changes to existing ones. This is not the case with the current state of the art.

Support for the course creator is limited in the courseware validation state of the art. Little research has taken place looking at how to represent the courseware construction concerns. The state of the art concentrates on allowing the course creator to define validation criteria for only a pre-defined types of courseware problems. Just one approach (ontology-based guidance) considers the possibility of separating the validation criteria from the validation framework programming logic, allowing the validation criteria to be exposed and determined by the course creator. This is significant as we have discovered in the CoCoA literature that many of the course creators that used the CoCoA tool found that the tool invalidated what they considered valid courseware (i.e. the course creators did not agree with the valid courseware definition of the CoCoA developers). We have therefore deemed it an important requirement for courseware validation to empower the course creator by allowing them to define what they consider to be valid and invalid in courseware.

We also find that the type of courseware that can be validated using the state of the art is somewhat limited. The approach taken by the majority of validation state of the art is to simulate a learner's behaviour in courseware. This works effectively on simple, non-adaptive courseware. As courseware gets bigger and more complex, allowing for personalisation, this approach runs into complexity issues. To effectively validate personalisable courseware an alternative approach to simulating learner behaviour must be established.

3.5 Chapter Conclusion

In this chapter, we reviewed the state of the art in courseware construction and then looked in detail at current courseware validation approaches. The chapter has outlined the context within which we position our research.

The majority of courseware construction tools provide interoperability with a courseware delivery environment through the ADL SCORM or IMS LD specification. This has caused the majority of courseware authoring approaches to be at the same granularity level as these TEL specifications, between the "course" and "lesson" granularity level. It should also be noted that there is a wide variety of internal modelling paradigms used by courseware construction approaches. Many base their internal data model on a TEL specification but some have developed custom internal modelling paradigms. This motivates positioning a courseware validation in line with the granularity level the TEL specifications operate at.

For our research to have maximum impact, it must be able to interoperate using the TEL specifications, but also be able to change with the TEL specifications as they mature, or even as new ones are developed.

We have presented four approaches to validation. Three of the approaches (logic-based, CoCoA and trap detection) are based on simulating a learners progression through courseware. The logic-based approach uses planning, CoCoA uses a tree traversal algorithm and the trap detection approach uses petri nets. There are scalability issues associated with these approaches and as courseware gets more complex, with personalisation becoming the norm, we believe this to be a serious limiting factor for these approaches. The validation approaches addressed in this chapter, with the exception of ontology-based guidance, are also somewhat limited in what they can validate in courseware. The validation scope of these approaches is pre-defined. This motivates the need for an extensible validation approach, one where the course creator can define the validation criteria, by which courseware is validated against. Ontology-based guidance (section 3.3.3) could allow for the course creator to define a greater variety of validation criteria. Ontology-based guidance is primarily focused on the use of templates and guidelines to help the course creator to create better courseware. The template approach exposes the validation criteria in the form of SWRL rules allowing the course creator to define templates based on instructional design theory. This could be extended to define all validation criteria.

Chapter 4

Defining a Courseware Validation Framework

4.1 Introduction

In this chapter we provide an outline of our courseware validation framework. In section 4.2 we introduce the actors involved in courseware validation, then in section 4.3 we look at the courseware construction concerns and finally, in section 4.4, we will outline our validation approach.

The courseware construction concerns, in section 4.3, are concerns that must be addressed by the course creator when constructing courseware. Our research looks to automatically ensure that the course creator has addressed these concerns correctly. We classify courseware validation problems using these concerns.

After outlining the courseware construction concerns we will outline the focus of our validation framework in section 4.4. This section gives a comprehensive overview of what a valid courseware is and how our validation approach can check if a given courseware is valid or invalid.

4.2 Courseware Actors

There are many actors involved in the construction and delivery of courseware. Each of these actors define a role that one, or many, persons can fill. One person can also fulfil many roles. In figure 4.1, we have outlined the main roles in the courseware construction process.

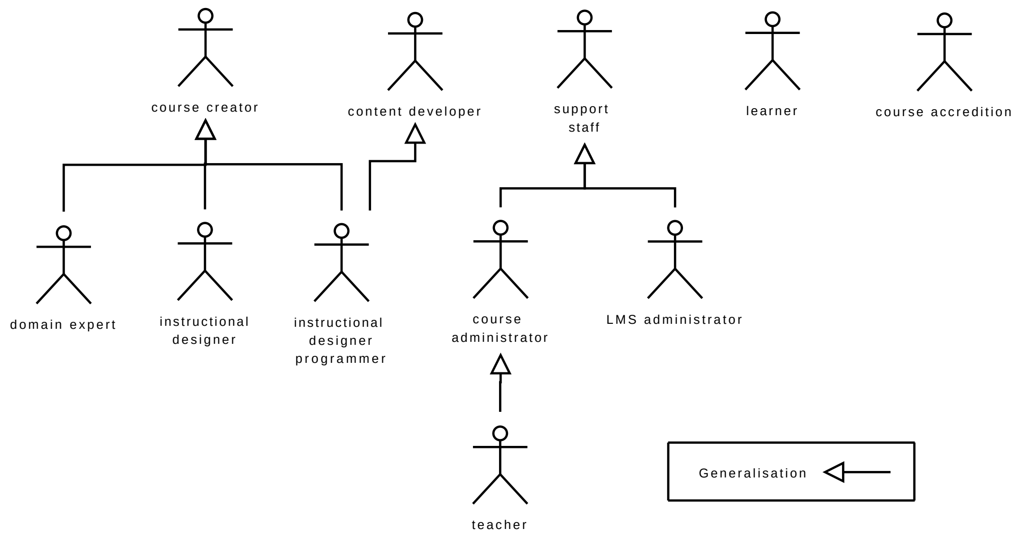


Figure 4.1: Courseware actors

There are five key roles in the courseware life-cycle, the *course creator*, the *content developer*, *support staff*, the *learner* and *course accreditation*. The *course creator* creates the courseware package, he or she is responsible for deciding what content to put into the courseware and the instructional design used in the courseware to determine how the LOs are delivered to the *learner*. We can divide the *course creator* role into three key sub-roles, *domain expert* - who is an expert in the subject domain to be taught, *instructional designer* - who is an expert in how courseware concepts should be taught in a given context and *instructional designer programmer* - a technical role that encodes instructional design using a courseware specification. The *content developer* is responsible for creating the learning content, such as slideshows, simulations, interactive media, podcasts, and assessments. An *instructional designer programmer* can also be a *content developer*, for example defining sequencing logic in a LO. *Support staff* are staff who support the *learner* through the learning process. The *learner* represents the individuals taking the courseware, who have a training need. *Learners* can be grouped into stereotypes where all members of the stereotype have a common training need or some learning characteristic in common. The *course accreditation* role, is generally the organisation or entity that accredits a course and specifies the scope of the courseware.

We note that the principle actor involved in courseware validation is the *course creator*. The *course creator* will design the courseware in accordance to the courseware scope as defined by the *course accreditation* actor, and compose the courseware using content created by the *content developer* actor.

4.3 Courseware Construction Concerns

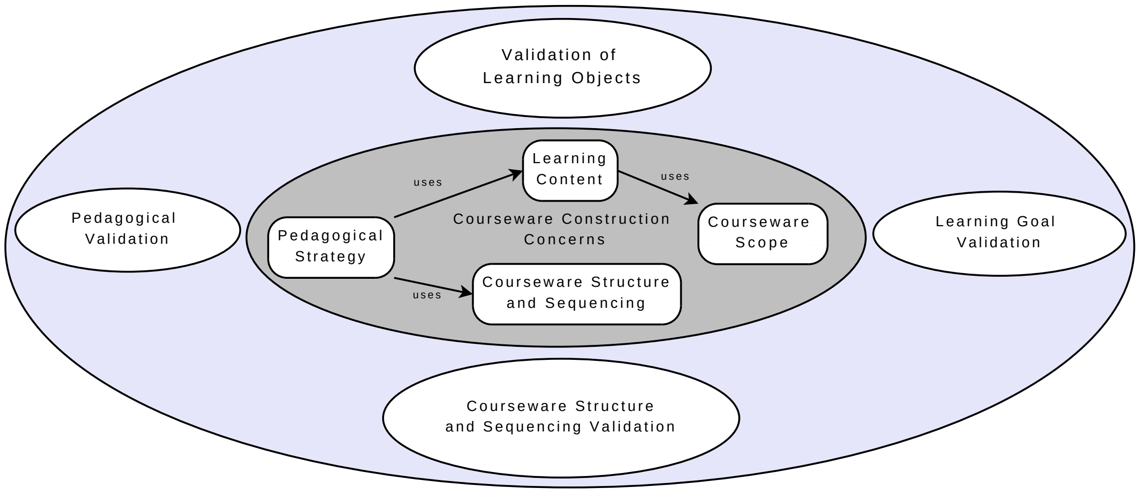

Validation of Learning Objects
Learning Content
uses
uses
Pedagogical Validation
Pedagogical Strategy
Courseware Construction Concerns
Courseware Scope
Learning Goal Validation
uses
Courseware Structure and Sequencing
Courseware Structure and Sequencing Validation

Figure 4.2: Courseware construction and validation concerns

In this section, we outline each of the courseware construction concerns that a course creator must consider during courseware construction. We use these concerns to derive possible courseware validation concerns, as illustrated in figure 4.2. As outlined in section 2.2 there are four principle courseware construction concerns; courseware scope, courseware structure and sequencing, learning content and pedagogical strategy. We can refine each of these courseware construction concerns in terms of questions that can serve as a basis for validation, producing courseware validation concerns. We can also map each validation concern to a courseware actor(s), as follows:

- **Courseware Scope -** This concern relates to whether the courseware will allow the learners to achieve the learning goals as defined by the course accreditation actor and also that the courseware is designed with the correct level of assumed knowledge. The courseware scope is outlined in detail in section 2.2.1. The concepts that need to be covered in order for the course scope to be satisfied are defined by a domain expert. The course accreditation actor defines the course scope. Examples of this concern are:
  - Are each of the learner's learning goals met?
  - What initial knowledge does the courseware assume, is this correct for the anticipated learners?
  - Does the courseware fill the gap in knowledge from the assumed initial knowledge to the learning goal for all learners?
- **Courseware Structure and Sequencing -** Looks at how courseware is structured and sequenced, in terms of the micro and macro level, as outlined in section 2.2.2. The courseware structure and sequencing is generally based on pedagogical principles and is defined by an instructional designer to be implemented in courseware by the instructional designer programmer. Examples of courseware structure and sequencing concerns are as follows:
  - Is the courseware consistent in terms of the size of its components?
  - How are courses, lessons and modules in the courseware structured?
  - Should concepts covered in some courseware component be taught before other concepts (pre-requisites between concepts)?
  - How should LOs in a lesson, which teach a particular concept to the learner, be delivered?

  - Does the effectiveness of specific LOs depend on some other LO or, more generally, a concept being covered first, and if so, is this considered?

- **Learning Content in Courseware -** Here the course creator is concerned with the appropriateness of the LOs in courseware used to teach a concept to the learner. LOs are developed by content developers. LOs in courseware can be evaluated along various dimensions such as:

  - Content Relevancy - Is the content the most recent version and a good fit for the micro-level courseware structure.
  - Technical Issues - Is the format of the content appropriate? Is the size of the content appropriate (e.g. broadband availability could be an issue for the learner).
  - Pedagogical Considerations - this considers issues to do with the suitability of the content for the learner (e.g. are particular learning styles facilitated) in a LO.

  Examples of the types of learning content validation criteria the course creator is concerned with are as follows:

  - Is the LO placed at the most suitable place in the courseware?
  - Does the content allow the learner to attain the course learning goals adequately and does the assessment content test the learner to ensure the learning goals have been met?
  - Is the content format correct for the expected courseware delivery environment?
  - Is the content format correct for the expected learners?
  - Is the content used suitable?

- **Pedagogical Strategy Used -** This concern is about the approach taken by the course creator to allow learning to take place, as outlined in section 2.2.4. The domain expert and instructional designer decide on what pedagogical strategy to use. Examples of validation criteria that this concern is associated with are:

  - Does the courseware design consider all possible learner profiles expected to take this course (courseware personalisation)?
  - What learning styles does the courseware accommodate?
  - Does the courseware apply an instructional design correctly?

The courseware construction concerns are not completely independent of each other. For example in defining the pedagogical strategy courseware construction concerns, courseware structure and sequencing concerns could be used.

4.4 Validation Focus of our Research

Here we define our courseware validation conceptual framework [Melia and Pahl, 2007c]. We begin by looking at how the course goal and structure can be validated. Following this we will look at how LO metadata, such as that found in IEEE LOM, can be used in validation.

The discussion in this section follows on from the courseware construction concerns identified in the previous section. In figure 4.2 we summarised the relationship between the courseware construction concern and the validation focus of our research. We have identified the following types of courseware validation, which we address in detail in this section:

- Course scope validation
- Courseware structure and sequencing validation
- Validation of Learning Objects in courseware
- Pedagogical validation

We note that the courseware validation concerns we address in this thesis can and do have relationships between each other. For example courseware sequencing problems that do not allow the course goal to be achieved will cause a scope and sequencing validation problem. This is known as a cross validation concern, where one type of courseware problem causes another. In this thesis we do not address the cross validation concerns.

Our approach to courseware validation is knowledge-based. Knowledge-based courseware validation uses a conceptual knowledge structure as a common point of reference for expressing validation specifications for a given courseware definition. Knowledge is captured in the form of a subject domain model, where a domain model defines domain concepts and relationships between domain concepts. The use of a subject domain model as a common point of reference in validation means that the domain model becomes a focal point in validation, as validation concerns can be specified in terms of the domain model.

4.4.1 Course Scope Validation

Course scope validation ensures that courseware developed satisfies the course's defined learning goals and also assumes the correct knowledge for the anticipated learners.

In order to verify that courseware satisfies each learner's course learning goals, it is necessary to define the course goal in terms of the courseware. This can be done by specifying courseware and the course goals in terms of a domain model, where the domain concept model captures concepts covered by the courseware and defines concepts as learning goals for learners.

In section 2.2.1.1, we outlined two broad knowledge types that can be used when specifying course goals and a numeric value for knowledge level. Using a concept domain model as described above does not allow for the validation of the knowledge type or level. This dynamic of knowledge must also be modelled when describing the courseware elements and the course scope details.

4.4.2 Courseware Structure and Sequencing Validation

LOs used in a given courseware are annotated with metadata. This metadata can reference domain model concepts, from a external knowledge classification system. From this we can infer what concepts are covered in courseware, where they are covered, and by which LOs. We can validate courseware structure in two ways by doing this, firstly intraconceptually looking at how LOs that cover the same concept are structured and sequenced, and inter-conceptually looking at how concepts in the courseware are composed and sequenced. Intra-conceptual courseware structure validation evaluates micro-level courseware structure, while inter-conceptual courseware structure addresses macro-level course structure.

4.4.2.1 Intra-conceptual Courseware Structure

Intra-conceptual courseware structure is concerned with the micro-level courseware structure, as described in section 2.2.2. Its main objective is ensuring that all potential sequences of LOs covering a particular concept in courseware is instructionally sound, as defined by the course creator.

The intra-conceptual courseware structure can be designed using a "planning sheet", outlined by Gagné et al. in [Gagné et al., 2005, ch12]. Using a planning sheet the course creator can specify a blue print for teaching a courseware concept, by specifying the type of LOs needed to teach a concept and also desirable LO type sequences. The planning sheet can be seen as an effective way for the course creator to define a default structure and sequence for teaching a courseware lesson.

We investigate how a intra-conceptual courseware structure and sequencing strategy can be defined, such as that found in a planning sheet, and used to validate a given courseware, in section 6.5.

4.4.2.2 Inter-conceptual Courseware Structure

Inter-conceptual courseware structure is concerned with how a learner moves from one course concept to another. Inter-conceptual courseware structure specifications should not be prescriptive, in that only the inter-conceptual sequences described by the courseware structure are valid. Instead inter-conceptual courseware structure should use constraints to indicate undesirable conceptual courseware sequences.

Constraints can be expressed in two ways; the course creator can specify explicit sequencing constraints between concepts or domain model constructs can be used to specify implicit sequencing constraints between two concepts. Typically explicit sequencing constraints are defined by the course creator using a modelling construct to indicate that one concept is a pre-requisite for understanding another concept. Implicit sequencing constraints can be specified based on a domain model structure.

4.4.3 Validation of Learning Objects in Courseware

The IEEE LOM [IEEE LTSC, 2002] allows for the automated analysis of what LOs are used in courseware and how they are used. Our approach is to evaluate a LO's suitability for its position in the courseware structure. To use LOM to validate a LO, we must examine the types of validation that can be carried out on each LOM attribute. As LOM has an extensive collection of attributes to describe a LO, we group attributes with similar ranges. Our motivation for doing this is that a validation approach based on a LOM attribute with a particular range can be generalised to any LOM attribute with the same range. We distinguish between three groupings of LOM attributes:

- Simple LOM attributes.
- LOM attributes which reference external resources.
- LOM attributes which classify a LO according to some external taxonomy.

Simple LOM attributes are those attributes, that can be easily compared to a fixed value or another LOM attribute value. These include any attribute with an atomic numeric or pseudo-numeric range. An example of such an attribute is the "interactivity level" attribute of LOM, which has a range of {very low, low, medium, high, very high}. These LOM attributes allow for simple rules to be specified based on comparing one LO to another, or comparing LOM attribute values with explicit values.

LOs often exist as part of some bigger educational entity. To allow for this LOM has attributes that reference external resources, such as other LOs. The *relation* section of LOM allows a LO to reference other LOs. The types of references include; *basedOn*, *hasPartOf* and *isRequiredBy*. The course creator can choose to ignore when a LO references another resource or may use it in validation, for example, ensuring that there is a sequencing constraint between courseware components that use LOs that are related through a *basedOn* relationship.

LOM can use an external classification systems to catalogue a LO according to some external taxonomy (e.g. The ACM Computer Classification System [ACM, 1998]). A successful implementation of this can be seen in the European Schoolnet project, which uses the classification section of LOM to point to an external competency taxonomy [VanAssche, 2007, Sicilia, 2005]. The LOM classification section allows LOs to be classified along any of the following dimensions; *prerequisite, educational objective, discipline*. An external classification system allows us to plot a given LO in terms of a conceptual knowledge structure, allowing LOs to be linked to a domain model and transitively linking courseware as a whole. Validation rules can then be specified in terms of a LO relation with the external classification system.

LOM attributes must also be validated in terms of their surrounding courseware context. For example, validation may ensure that the sum of the duration time of all LOs in a given courseware does not exceed a certain threshold. This type of validation is also important to ensure a sense of uniformity in the courseware, for example, ensuring the semantic density of each consecutive LO delivered to the learner remains the same or gets progressively larger or smaller.

4.4.4 Pedagogical Validation

The pedagogical strategy for courseware defines how the courseware is designed to allow learning to take place. We define pedagogical strategy in the context of courseware in section 2.2.4. In validation, we aim to ensure that the pedagogical approach that the course creator looks to use has been implemented correctly. In our validation approach we have divided this into two core categories; validation of the implementation of an instructional design in courseware and the validation of the application of personalisation.

Instructional design defines the probabilistic best approach for learning given a certain pedagogical context. The instructional design used in courseware could be an implementation of an instructional design theory from the literature [Gagné et al., 2005, Reigeluth, 1999a, Reigeluth, 1983a, Briggs et al., 1991] or could be his or her own instructional design. Validation should allow the course creator to ensure that the instructional design theory he or she is using has been implemented correctly in the courseware definition.

Personalisation allows the delivery of courseware that adapts to an individual learner's, or group of learner's (stereotype), learning needs. A courseware definition must define how adaptive behaviour takes place. Courseware validation should allow the course creator to check that adaptive behaviour has been defined correctly and will be implemented as anticipated for each learner instance.

4.5 Chapter Conclusion

Our aim in this chapter was to define a conceptual framework for courseware validation. To do this we identified the principle actors involved in courseware construction. We then outlined the courseware construction concerns that the course creator must consider when constructing courseware. These concerns allowed us to define the overall scope of courseware validation for this thesis.

From the courseware validation definition in section 4.4 we define courseware as valid if the following conditions are satisfied:

1. Courseware satisfies the course learning goal for all learners taking the courseware (section 4.4.1).
2. The courseware structure and sequencing at the micro and macro level is sound (section 4.4.2).
3. Appropriate LOs are used for the courseware (e.g. format is correct, LO scope is adequate) (section 4.4.3).
4. The right LOs are delivered to the right learners at the right time (section 4.4.3 and 4.4.4).
5. Instructional design is applied correctly (section 4.4.4).

6. Adaptive strategies are applied to the courseware correctly (section 4.4.4).

To deal with the validation concerns outlined in this chapter we propose a Domain Specific Modelling Language (DSML) in chapter 5 to capture the courseware data available at the post-construction/pre-delivery stage of the courseware life-cycle. In chapter 6 a constraints-based validation language is defined in the context of our DSML to allow for the validation of courseware as outlined in this chapter.

Chapter 5

Defining the CAVIAr Data Models

5.1 Introduction

In this chapter, we will define the **C**ourseware **A**uthoring **V**alidation **I**nformation **Ar**chitecture (CAVIAr). CAVIAr is a set of models that have been comprehensively defined by the author to capture courseware and its validation criteria [Melia and Pahl, 2009, Melia and Pahl, 2007b, Melia and Pahl, 2007a]. We can divide the CAVIAr model into two parts the CAVIAr data models, which capture the courseware definition and its construction concern data, and the Validation Model, which defines what is correct and incorrect in courseware. The CAVIAr data models are as follows:

- Domain Model
- Learning Context Model
- Learning Resource Model
- Courseware Model

Figure 5.1 outlines the CAVIAr models using UML. The main CAVIAr models are the Learning Context Model and the Courseware Model. The Learning Context Model contains a domain model and the Courseware Model references the Learning Resource Model. CAVIAr model names will start with capital letters for the rest of this thesis.

The Domain Model captures the structure of curriculum knowledge. The learning context is defined in terms of the Domain Model and outlines courseware construction concerns, such as conceptual sequencing constraints and the course scope. The Learning Resource Model contains a representation of the learning resources used in the courseware. The Courseware Model defines the courseware that will be delivered to learners in a TEL specification-neutral way. Courseware elements refer to learning resources in the Learning Resource Model. LOs in the Learning Resource Model can be classified using a Domain Model concept. This is illustrated in figure 5.1 through the reference from the Learning Resource Model to the Domain Model.

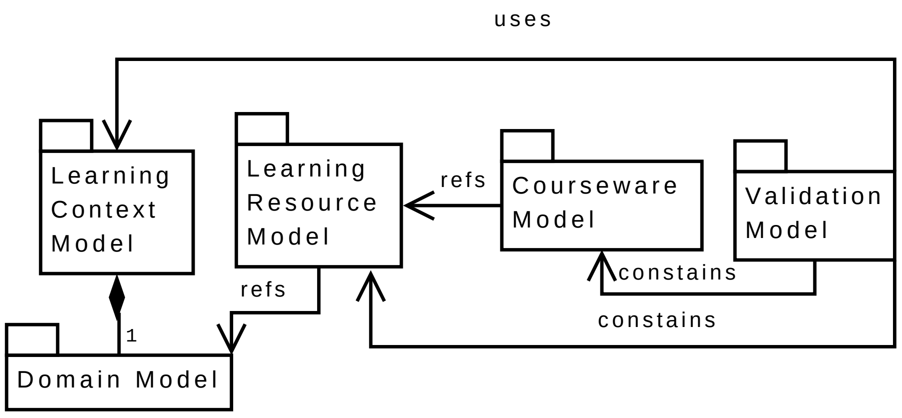


Figure 5.1: Overview of the CAVIAr Models

The Validation Model is defined by constraining the allowable structure of the Courseware Model and Learning Resource Model. To do this a constraint language is used to express constraints on the Courseware Model's and Learning Resource Model's abstract syntax definition, defining what is a valid Courseware Model definition. Constraints can be defined based on the Learning Context Model. The Courseware Model is covered in detail in chapter 6. In this chapter we will concentrate on defining the CAVIAr data models.

In order to define the CAVIAr data models, we look at how language notation and semantics are defined in section 5.2. This is then used to frame the definition for each CAVIAr data model discussed in the subsequent sections.

5.2 Language Notation, Syntax and Semantics

In this chapter we define a Domain Specific Modelling Language (DSML) for the purposes of courseware validation. DSMLs are a form of a Domain Specific Language (DSL) that can be defined and represented using graphical models. DS(M)Ls differ from General Purpose Languages (GPL) in that they can offer domain-specific notations, constructs and abstractions [Mernik et al., 2005]. An example of a GPL would be Java. As courseware construction is a highly specialised task, we have chosen to design a DSML for this purpose. We have been guided in this decision by the work of Mernik et. al, as we believe that a DSML offers the following advantages [Mernik et al., 2005]:

- Improved software economics by providing a modelling language that captures the dynamics of TEL and courseware development.
- Providing for a modelling environment which is familiar to the end-user course creator, empowering the course creator.

This language will be used to define the constructed courseware and its courseware construction concerns.

For language to be adequately defined it must consist of three parts [Harel and Rumpe, 2004]:

- Syntax
- Semantic Domain
- Semantic Mapping

Harel and Rumpe note two main forms of language, textual and diagrammatic. A textual language's basic syntactic expressions consist of linear character sequences making up words, sentences, paragraphs and so on. Diagrammatic language's basic syntactic expressions consist of lines, arrow, boxes and so on, and composition mechanisms including partitioning and connectivity. In order for correct interpretation of a language there must be a rigid syntax. An abstract syntax defines the structure and allowable elements in a language's concrete syntax, that is used by the end-users of a language.

A language's semantic domain provides the meaning for each syntactic expression. The semantic domain and syntax of a language are two separate entities; it is possible that many different syntax can specify the same semantics. The definition of a semantic domain can be expressed in a variety of ways from formal definitions to natural language.

A language's semantic mapping relates the language's syntax to a semantic domain. A common approach to providing a syntax with a semantic domain is to map it to another language with a well-defined semantic domain. The language's semantics are then defined transitively [Kurtev et al., 2006].

In the following sections, each of the CAVIAr data models are introduced by outlining its purpose. We then define an abstract syntax for the model. The model's abstract syntax is defined as a metamodel using MOF [OMG, 2003a]. MOF is outlined in section 2.3.1. We decided to use MOF to define the CAVIAr data models as it offered the most appropriate semantic expressivity needed for describing courseware validation concerns, as outlined in section 2.3. A graph-based semantic domain is then defined for the CAVIAr model being defined, which is then mapped to the abstract syntax. The CAVIAr semantics are defined to eliminate ambiguity in the language. Strict semantics will allow for algorithms to be based on the CAVIAr models. We have used a graph-based notation to define the semantic domain as it is an established approach for defining adaptive courseware languages [Vassileva and Deters, 1998, Karampiperis and Sampson, 2004, Cristea and Aroya, 2002]. Using graph-based semantics allows for easy comparison with such languages.

Languages may also have one or more concrete syntax. The concrete syntax is used directly by the end user and is defined in the confines of the abstract syntax. To complete the definition of the CAVIAr model's a "candidate" concrete syntax is defined. We use the term candidate as it is a suggestion on how to express the CAVIAr models, other concrete syntax could be specified. The candidate concrete syntax, outlined in this chapter, are used throughout this thesis. For CAVIAr to be useful it must be able to interoperate with a variety of existing TEL standards and specifications. To this effect we will also outline how the various CAVIAr models can be mapped to popular TEL standards and specifications, where appropriate.

5.3 The Domain Model

The purpose of the Domain Model is to represent the subject knowledge to be conveyed to the learner during courseware delivery. Domain models are used extensively in courseware and ITS construction. Murray outlines four main motivations for using explicit domain models as follows [Murray, 2003]:

- To model curriculum knowledge and structure - Allow for visualisation of conceptual curriculum elements, where a conceptual curriculum element is the conceptual breakdown of a course, but not actual learning material (e.g. topics, lessons, concepts).
- To model simulations of the world - Modelling of physical processes which allow for learning.
- To model expert knowledge - Modelling of several types of knowledge including problem solving expertise, procedural skills, concepts and facts.
- To model domain knowledge types - Modelling of the knowledge types coming together, for example, modelling the different knowledge dimensions (e.g. what, how, where) of a particular concept.

The use of domain models has become even more relevant as ontologies are used in a variety of scenarios, such as annotating and organising web resources on the Semantic Web, or as a basis for inferring knowledge in bioinformatics [Doniger et al., 2003].

The sole purpose of the Domain Model in CAVIAr is to express the structure of curriculum knowledge structure that courseware covers. The Domain Model does not need to be a comprehensive knowledge source for the subject domain but just a knowledge structure that is intuitive to the non-technical course creator that describes concepts and common relationships between concepts. This was one of the deciding factors when deciding the expressivity required for our research in terms of Daconta's ontology spectrum [Daconta et al., 2003], as outlined in section 2.3. It is worth noting that all knowledge in the Domain Model does not have to be covered by the courseware. The knowledge covered by the courseware may be a subset of the knowledge in the Domain Model.

5.3.1 Defining the Domain Model Abstract Syntax

The Domain Model is a formalism of knowledge defined as a concept graph, where a node represents a concept and an edge represents the relationship between two concepts. The CAVIAr Domain Model is pedagogically neutral, in that it does not prescribe any teaching or learning methodology over the knowledge it contains. This allows for the use of domain models that have been defined for some purpose other than instruction to be used in CAVIAr.

A concept graph for domain modelling in AEH authoring is formally defined by Cristea and deMooij in [Cristea and de Mooij, 2003]. In AEH the domain model serves as the main navigational tool, in which learning content is embedded. In courseware the domain model's use is most prominent as a semantic point of reference for LO annotation. An example of this can be seen in the IEEE LOM standard where the "classification" metadata is used to annotate a given LO using an external classification system (see section 2.2.3).

We define the Domain Model in the modelling technical space using a MOF-based definition. In figure 5.2, we outline the abstract syntax for the CAVIAr Domain Model using MOF. It defines the *Concept* as the primary building block of the *DomainModel*. A *Concept* has one attribute, the *name* of the concept. The concept *name* will uniquely identify the *concept* within a *DomainModel*. A *concept* can have many *Synonyms*, a *synonym* is another name by which a *concept* is known. This can also be used for labelling the *concept* with a *name* using an alternative language.

Concepts are related to other *concepts* by a *ConceptRelationship*. A *ConceptRelationship* has a direction from its *source* concept to its *target* concept. The *ConceptRelationship* has one attribute, *type*. The *type* defines the semantics of the relationship. There are two types of concept relationships, defined in the enumeration *ConceptRelationshipType*:

- *NARROWER* - The source concept has a broader semantic scope than the target concept. This allows for a taxonomic relationship between concepts.
- *RELATED* - There is a semantic relationship between two concepts. This relationship is symmetric.

The MOF defined abstract syntax is further constrained as follows:

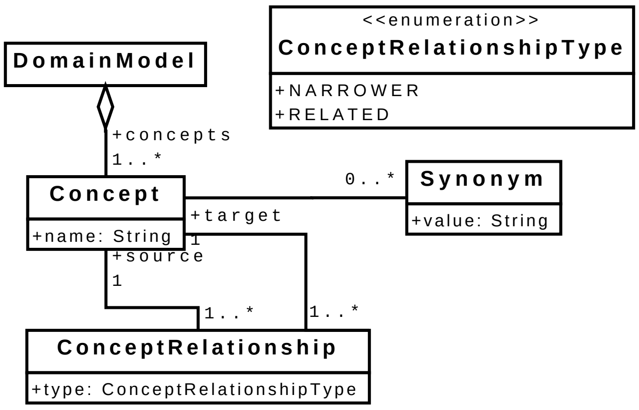


Figure 5.2: Domain model abstract syntax defined in MOF

- All concepts must have a name.
- All concepts must have a unique name.
- All concept synonyms must have a value.

These constraints are defined formally in OCL below in listing 5.1.

Listing 5.1: Domain model abstract syntax constraints defined in OCL

```
context Concept
    inv concept_name_must_have_value: name <> ''
    inv concept_must_have_unique_name: Concept.allInstances()->any(name <> self.name)
context Synonym
    inv synonyms_must_have_value: value <> ''
```

5.3.2 Defining the Domain Model Semantics

A concept graph CG is defined as $CG \subseteq C \times E$ where C is a set of concepts and E is a set of edges.

- A concept $c \in C$ is described with a set of names $c = \{n_1, n_2, ...n_k\}$, where the first name in the set shall be the default concept name. Names in the same set are synonyms of each other. Each concept is uniquely identifiable by its default concept name.

- An edge $e \in E$ is a tuple $<c1, c2, t_e>$ that relates two domain concepts, $c1 \in C$ and $c2 \in C$. The edge type, t_e, defines the semantics of the edge. We define two types of edges allowing for simple ontologies to be defined:
  - *NARROWER* - Allowing for a semantic taxonomy working from the concrete to the more abstract concepts.
  - *RELATED* - Allows for the grouping of concepts, that one concept is related to another concept.

In order to complete the semantic definition of the Domain Model, we provide semantic mappings between the domain model's abstract syntax and its semantic domain. The concept graph defined as *CG* is mapped to the *DomainModel* in the abstract syntax. The MOF-defined *Concept* is mapped to the set C in the tuple-based domain model definition, *ConceptRelationship* is mapped to the set of edges E. The first *name* in c is the default name for c, where $c \in C$ is mapped to the name attribute of a MOF-defined *Concept* class, while the concept's *Synonyms* are the remainder of the concept names in the set of c. The taxonomic relationship and the related relationship in CAVIAr abstract syntax can be directly mapped to the taxonomic and related relationship in the semantic domain.

5.3.3 Domain Model Interoperability

For the CAVIAr Domain Model to be useful, it must be interoperable with the state of the art in domain model definitions. To illustrate interoperability at a conceptual level we define mappings to the well-defined Simple Knowledge Organisation Structure (SKOS) standard [Miles and Brickley, 2005].

SKOS is defined using OWL [W3C, 2004]. SKOS is a well-defined candidate recommendation from the World Wide Web Consortium (W3C), used to express basic conceptual structures and schemata. SKOS therefore operates in the ontological technical spaces, whereas CAVIAr operates in the metamodelling technical space. Mappings between these two technical space can be seen in the Ontology Definition Metamodel (ODM), where OWL classes are defined as MOF classes [Gašević et al., 2006].

The primary component of a SKOS ontology is the *skos:concept*, which is an instance of *owl:class*. Concepts are related by two types of semantic relationships; associative relationships - *skos:related*, and taxonomic - *skos:narrower* and *skos:broader*. The *skos:narrower* and *skos:broader* relationships are used to define when a *concept* has a narrower or broader scope than another.

SKOS can be mapped to the CAVIAr Domain Model abstract syntax as follows:

- CAVIAr *DomainModel* is mapped to the RDF graph which hosts a SKOS *ontology*
- CAVIAr *Concept* is mapped to the *skos:concept*.
- CAVIAr *ConceptRelationship* of type *NARROWER* is mapped to SKOS *skos:narrower* relationships, while the inverse of a *NARROWER* relationship from the target to the source can be mapped to the *skos:broader* SKOS relationship
- CAVIAr *ConceptRelationship* of type *RELATED* is mapped to the SKOS relationship *skos:related*
- The set of names for a given concept, defined as synonyms and as the concept name attribute, can be mapped to the *ThesaurusTerm* in the SKOS definition. Those terms that are related to the *skos:concept* through the *skos:altLabel* are a *Synonym* in the CAVIAr Domain Model, while the *skos:prefLabel* is used to determine the CAVIAr concept's *name* attribute.

5.3.4 Defining a Candidate Concrete Syntax Definition

We propose a simple graphical concrete syntax for the CAVIAr Domain Model. We do this to define a modelling language that is intuitive enough to be used by a non-technical end user, such as a course creator. In defining the concrete syntax we will make reference to the abstract syntax defined using MOF.

A *Concept* is represented as an ellipse, the value of the concept's *name* attribute occupies the centre of the ellipse. *Synonyms* are represented in the domain model as broken line ellipses, where the *value* attribute of the *synonym* occupies the centre of the ellipse. *Synonyms* are related to *concepts* by a solid black line.

The two types of concept relationships are represented as follows:

- *NARROWER* - Broken-line arrow from broader concept to narrower concept. The arrow head is placed at the *ConceptRelationship* target concept that being the narrower concept.
- *RELATED* - Broken line between two related concepts

In figure 5.3, we outline an example of the Domain Model concrete syntax. This example is taken from the databases subject domain. In the example the *concept* named “SQL” has a *Synonym*, “Structured Query Language”. “SQL” is related to “Relational algebra” and has a broader semantic scope than “Views”, where the arrow indicates “Views” is *narrower* than “SQL”.

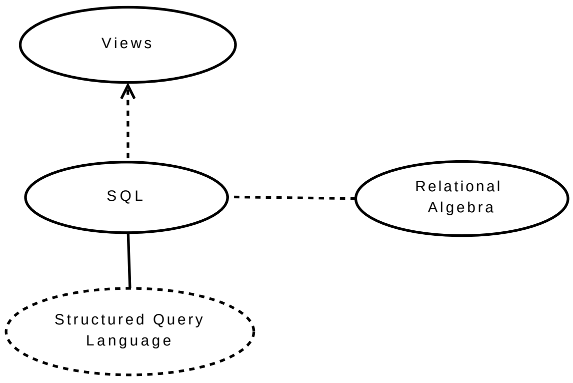


Figure 5.3: CAVIAr Domain Model concrete syntax example

5.4 The Learning Context Model

The Learning Context Model defines domain-related pedagogic information. The Learning Context Model extends the Domain Model, by defining courseware requirements as knowledge elements in terms of Domain Model concepts. We can describe the Learning Context

Model as layered on the Domain Model. For this reason a Domain Model must be included in the learning context definition. The Learning Context Model is made up of:

- Conceptual Sequencing Constraints - Defines where knowledge of one concept is necessary to understand another concept (see section 2.2.2.2).
- Learner Stereotype Definitions - Defines what a learner grouping needs to know to successfully complete the course and what a learner grouping is assumed to know prior to starting the course.

A learner stereotype definition allows the course creator to define learner groupings, defined in terms of the assumed initial knowledge of the learner grouping prior to starting the courseware and the learner grouping's learning goals. This is known as the course scope, where a course scope is defined for each learner stereotype (course scope is covered in detail in section 2.2.1). The course goal and assumed initial knowledge are defined as knowledge elements in terms of the Domain Model. A knowledge level, a knowledge type, and a Domain Model concept define knowledge elements.

The abstract syntax for the Learning Context Model is defined in the following subsection. This is then mapped to a graph-based semantic domain definition. The section concludes with a description of a candidate concrete syntax.

5.4.1 Defining the Learning Context Model abstract syntax

In figure 5.4 we outline the Learning Context Model abstract syntax in MOF. One of the central elements of the learning context is the *Concept* modelling element from the Domain Model as the learning context is defined in terms of the Domain Model.

In the Learning Context Model an additional concept relationship is defined - *ConceptPreReq*. This relationship allows the course creator to define where knowledge of some Domain Model concept is necessary to understand another Domain Model *concept*. A *ConceptPreReq* relationship defines a relationship between a *concept* and a *KnowledgeElement*, where the pre-requisite knowledge is defined as a *KnowledgeElement*. A *KnowledgeElement* is defined with knowledge *type*, a knowledge *level*, and also a reference to a Domain

Model *concept*.

The *LearningContext* is also made up of *LearnerStereotypes*. A *LearnerStereotype*, ls_i is defined by a *name* and a set of *KnowledgeConstraints*. The stereotype *name* uniquely identifies each stereotype. There are two types of knowledge constraints that can be defined on a learner stereotype, *Goal* and *PresumedKnowledge*. *PresumedKnowledge* defines the knowledge that the course creator believes the learner stereotype to have prior to starting the courseware. The *Goal* knowledge represents the knowledge state a learner should have after taking the courseware. The *PartOfRelationship* allows for the construction of composite goals. If a *Goal*, $g1$ is part of another *Goal*, $g2$, the *LearnerStereotype* has an overall goal, which is the union of the knowledge elements in the two *goal* sets, $g1 \cup g2$. Alternative goals are defined using the *AltGoal* construct, where an *AltGoal* instance is associated with learning context *goals* that are alternatives to each other.

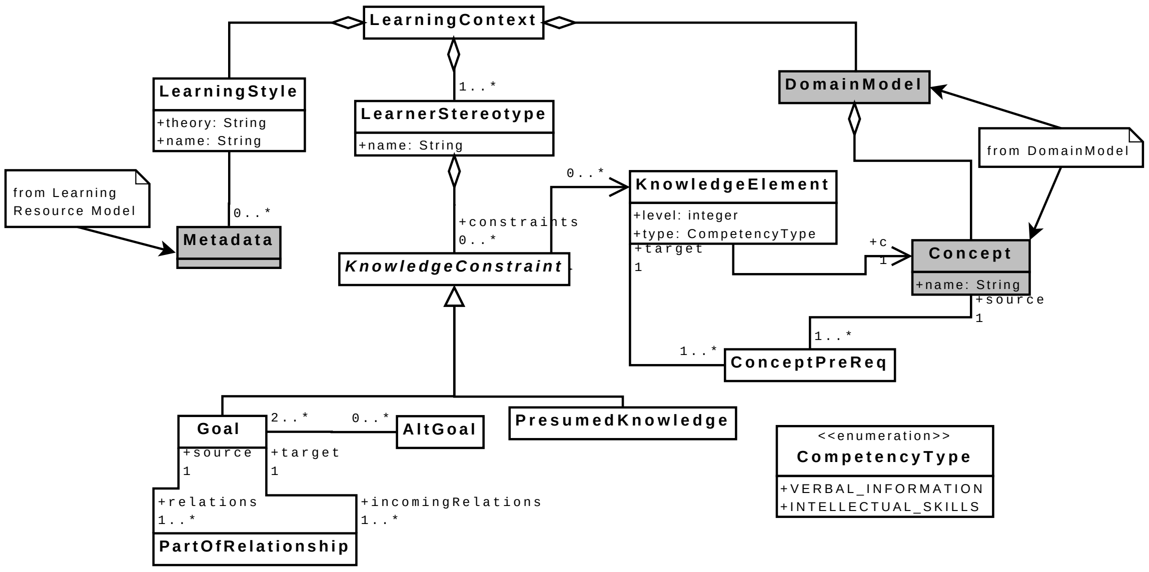


Figure 5.4: Learning context model abstract syntax

The Learning Context Model also allows the course creator to define the learning styles that he or she wants to use in validation. Learning styles can then be associated with LO metadata types. Capturing the anticipated learning styles that will be used in the courseware allows the Courseware Model to validate the course in terms of how learning styles are used in the courseware. This is a similar approach to the integration of learning styles into adaptive strategies defined by Stash et al. in [Stash et al., 2004].

The MOF defined abstract syntax is further constrained as follows:

- If a learner stereotype has a goal and presumed knowledge for knowledge elements that reference the same concept, the knowledge level of the goal must be greater than the knowledge level of the presumed knowledge.

These constraints are defined formally in OCL below in listing 5.2.

Listing 5.2: Learning Context model abstract syntax constraints defined in OCL

```
context LeanerStereotype
inv goal_knowledge_level_greater_than_presumed_knowledge:
--Get set of concepts that are goal and presumed knowledge concepts
let overlappingConstraints : Set(Concept) = self.constraints->select(oclIsTypeOf(Goal))
    ->iterate(i:KnowledgeConstraint; res:Set(KnowledgeElement)=Set{}| res->union(i.knowledgeElements))
    ->terate(j:KnowledgeElement; res2:Set(Concept)=Set{}| res2->union(i.concept))
    ->intersection(self.constraints->select(oclIsTypeOf(PresumedKnowledge))
    ->iterate(x:KnowledgeConstraint; res3:Set(KnowledgeElement)=Set{}| res3->union(x.knowledgeElements))
    ->terate(y:KnowledgeElement; res4:Set(Concept)=Set{}| res4->union(y.concept)))
in
     self.constraints->select(oclIsTypeOf(Goal))->collect(knowledgeElements)
    ->select(ke | overlappingConstraints->includes(ke.c))
    ->forAll(ke | ke.level >
         self.constraints->select(oclIsTypeOf(PresumedKnowledge))
        ->collect(knowledgeElements)->select(kk|kk.c = ke.c))->forAll(level > ke.level))
```

5.4.2 Defining the Learning Context Model Semantics

The Learning Context Model LCM is made up of the tuple $<P, LS, S>$, where P is is a set of directed edge between Domain Model concept nodes, which specify a pedagogical pre-requisite relationship between concepts. LS is a set of learner stereotype definitions and S is a set of learning style definitions

- The basic building block of the Learning Context Model is *KnowledgeElement*. A *KnowledgeElement* ke is defined by the tuple $<kl, kt, c>$, where *kl* is a knowledge level. Knowledge levels is a value between 0 and 1, where 0 indicates no knowledge of the concept and 1 indicates full knowledge of the concept, c. *kt* is a knowledge type, where a knowledge type can be of type verbal information or intellectual skill, the two learning outcomes, defined by Gagné et al. in [Gagné et al., 2005] that can be defined in terms of Domain Model concepts (outlined in section 2.2.1).

- A pre-requisite constraint $p \in P$ is described using the tuple $<c1, ke>$, where $c1 \in C$, and ke is a knowledge element. The learner must have achieved the knowledge defined in ke for the pre-requisite defined on c to be satisfied.

- A learner stereotype $ls \in LS$ is described by the tuple $<G, AK, n>$, where G is a set of learning goal knowledge elements for ls, AK is a set of assumed existing knowledge elements for ls and n is the name of the stereotype.

- A goal $g \in G$ is described by the tuple $<KE, alt, NG>$, where KE is a set of knowledge elements defining the learning goal, alt defines a set of alternative goals and NG is a set of nested goals within the goal g. Nested goals create a hierarchy of goal sets. The alt in a goal defines an alternative goal set for that goal, allowing for the definition of alternative learning goals.

- Assumed knowledge element, $ak \in AK$ is described by the tuple $<KE>$, where KE is a set of knowledge elements defining the presumed knowledge for this learner.

- A learning style $s \in S$ is defined by the tuple $<st, sn, M>$, where sn is the name of a learning style within the given theory st. The learning style is associated with a set of metadata, M, that identifies LOs that suit the learning style s.

We provide a semantic mapping for the Learning Context Model by mapping between the abstract syntax and the semantic domain. The *LearningContext* MOF class is mapped to the container, containing the sets P and LS, LCM. The pre-requisite set P is mapped to the MOF class *ConceptPreReq*. The learner stereotype definition LS is mapped to the *LearnerStereotype* MOF class. The MOF class *KnowledgeConstraint* defines learner relationships to knowledge, through the *KnowledgeElement* MOF class, which can be mapped to the set KE. The *KnowledgeConstraint* class is specialised into two MOF classes, *Goal* and *PresumedKnowledge*, which can be mapped to the sets G and AK respectively.

5.4.3 Learning Context Model Interoperability

The learning context model allows for the definition of the anticipated learner stereotypes in terms of knowledge elements. The state of the art provides for specifications describing learners and describing knowledge. Learners can be described using the IMS Learner Information Package (LIP) specification [IMS, 2005]. The learner stereotype definition can be mapped to the IMS LIP specification in the following ways:

- IMS LIP competency is mapped to learner stereotype presumed knowledge
- IMS LIP goal is mapped to learner stereotype goal

Knowledge can be represented in terms of a competency or educational objective using the IMS Reusable Definition of Competency or Educational Objective (RDCEO) specification [IMS, 2002]. A knowledge element or group of knowledge elements defined in the Learning Context Model can be mapped to the a RDCEO definition or group of RDCEO definitions.

5.4.4 Defining a Candidate Concrete Syntax Definition

We propose the following concrete syntax to assist in the communication of learning context definitions. We have defined the concrete syntax through an example, which is illustrated in figure 5.5.

A learner stereotype are depicted as a stick person, e.g. *stereotype1*. The stereotype is associated with the knowledge constraints that define this stereotype. Knowledge constraints are depicted as boxes, with instances of *PresumedKnowledge* denoted with a small line in the top right corner of the box. Knowledge constraints are associated with *concepts* in the Domain Model through a *KnowledgeElement*. The association between knowledge constraint and concept depicts a *KnowledgeElement*. The knowledge element *level* is depicted as a label on the relationship between the knowledge constraint and the concept, while the arrow head image depicts the type of knowledge, a single head indicates the knowledge type is *VERBAL_INFORMATION* and a double arrow head is *INTELLECTUAL_SKILLS*.

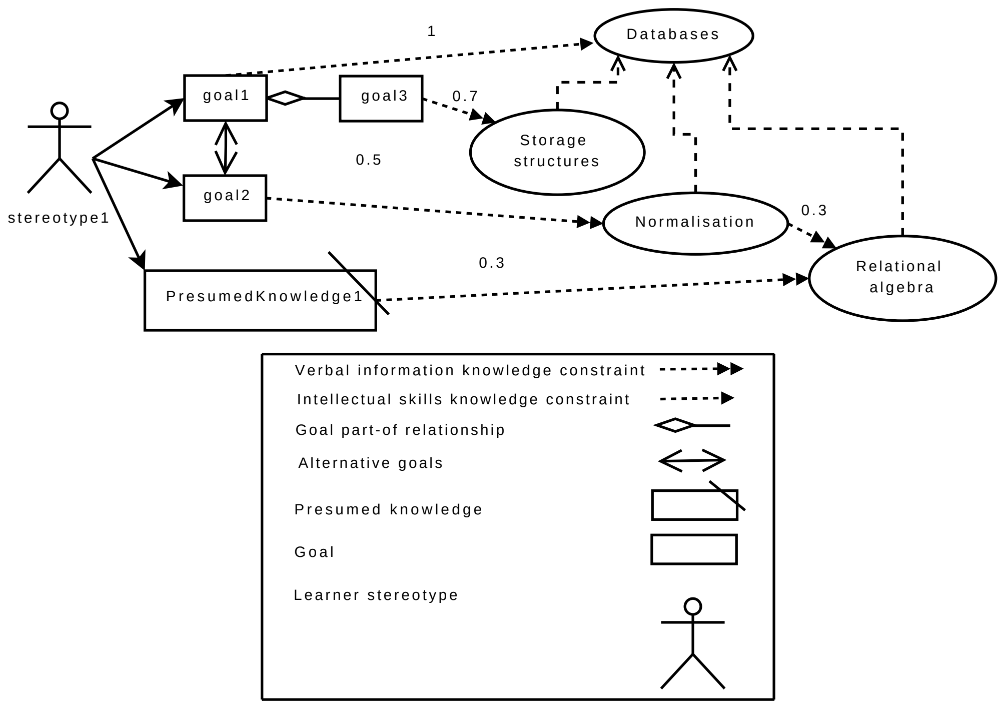


Figure 5.5: Learning Context concrete syntax example

For example, “goal3” has a knowledge element with a type of *INTELLECTUAL_SKILLS*, with a level of 0.7 in the “Databases” *concept*.

Alternative goals are denoted using a bidirectional arrow as is the case for “goal1” and “goal2”. A relationship with a diamond indicates that one *goal* is part of another *goal*, as shown in figure 5.5, where “goal3” is part of “goal1’.

A knowledge element association between two concepts is a pre-requisite relationship between those concepts. The *concept* at the arrow head end is the target of the *ConceptPreReq* relationship. In figure 5.5 “Relational algebra” is a pre-requisite of “Normalisation”.

5.5 The Learning Resource Model

The Learning Resource Model contains representations for learning resources used in courseware. The learning resource representation is based on the IEEE Learning Object Metadata (LOM) standard [IEEE LTSC, 2002], enabling a direct mapping from LOM to the Learning

Resource Model. This section outlines how learning resources are defined in CAVIAr.

5.5.1 Defining the Abstract Syntax

In figure 5.6 we outline the Learning Resource Model's abstract syntax in MOF, the data types used in figure 5.6 are in figure 5.7. A *LearningResourceModel* is composed of *Resources*. There are two types of *Resource*, *LO* and *Service*. Where a LO is as defined in section 2.2.6 and a service is as defined in the IMS LD literature as "any service that is needed during learning, e.g. communication services, search services, monitoring services, and collaboration services" [Koper and Olivier, 2004].

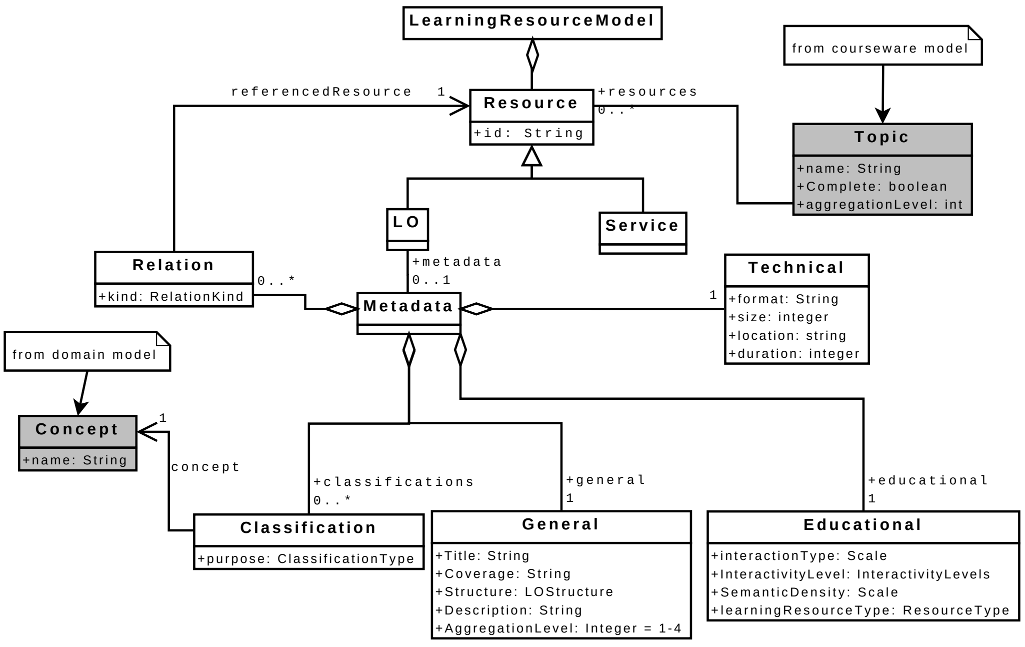


Figure 5.6: Learning resource model abstract syntax

The *LO* is associated with *Metadata*, which can be directly mapped to the IEEE LOM standard [IEEE LTSC, 2002]. The *Metadata* is composed of five parts - *Relation, Classification, General, Educational* and *Technical*. All modelling elements contained in *Metadata* have attributes that are used to describe the associated *LO*. The *Classification* associates the *LO* with a Domain Model concept for some *purpose*, where *purpose* is an attribute of the *Classification* class. The *Relation* metadata relates this LO to other resources. The LO

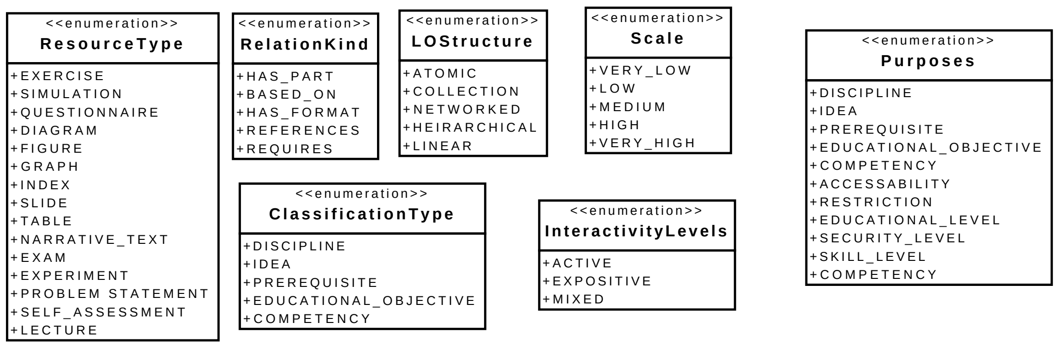


Figure 5.7: Learning resource model data types

relationship is defined by the *Relation* attribute, *kind*. The five *Metadata* components are derived from the metadata types outlined in the LOM standard.

5.5.2 Defining the Learning Resource Model Semantics

We define the semantics of the Learning Resource Model as follows:

- A Learning Resource Model, LRM, is a set of learning resources, LR.

- There are two sets of learning resources captured in the LRM, a set of learning objects LO and a set of learning services LS, therefore $LR = LO \cup LS$.

- A LO learning resource is further defined by the tuple $<REL, CLASS, A>$, where *REL* is a set of relationships, that relates a *LO* with another *LR*, *CLASS* is a set of conceptual classifications for this *LO*. A is a set of annotations for this LO.

- A relation $rel \in REL$ contains the tuple $<lr, type>$where $lr \in LR$ and the type of relation, which is defined as a text value.

- A classification $class \in CLASS$ is defined by the tuple $<c, pur>$, where $c \in C$ and *pur* defines the purpose of the conceptual classification as a text value.

- A learning object's annotation A_{lo} is a tuple $<att, val>$, where att is the name of the metadata attribute and val refers to its simple text value.

To map the abstract syntax to the semantics we define the abstract syntax in terms of the graph-based semantics. The *LearningResourceModel* MOF class is mapped to the set of learning resources defined as the LRM set. The Learning Resource Model contains instances of the *Resource* class. There are two types of learning resources defined in the abstract syntax, *LO* and *Service*, which correspond to the sets LO and LS respectively. The *Relation* metadata class is mapped to the set REL, while the *Classification* class is mapped to the set containing conceptual classifications $CLASS$. The *General*, *Technical* and *Educational* classes define a series of attributes and values, used to describe the LO and are mapped to the set A.

5.5.3 Learning Resource Model Interoperability

The CAVIAr Learning Resource Model must be able to interoperate with external learning resource descriptions and metadata specifications and standards. In this sub-section, we outline how this is achieved by mapping the Learning Resource Model to common standards and specifications in TEL.

There are two types of *Resource* in the *LearningResourceModel*, the *LO* and the *Service*. A Learning Object (*LO*) is a representation of some learning resource. LOs in the CAVIAr courseware specification can be mapped to the *learning object* definition in IMS LD and to the *Resource* in the SCORM specification. The CAVIAr *Service* can be mapped to the *Service* definition in IMS LD, which together with a set of LOs creates the IMS LD *environment*.

Each *LO* is defined in terms of metadata. Each element contained in the *Metadata* can be mapped directly to an element of the IEEE LOM standard, for example the CAVIAr *General* MOF class can be mapped to the general section of LOM, the *Educational* MOF class can be mapped to the “educational” section of LOM, and so on.

5.5.4 Defining a Candidate Concrete Syntax Definition

The Learning Resource Model is principally about annotating learning resources in courseware. This data is verbose in nature and therefore if presented in a diagrammatic notation with the other CAVIAr models could cause confusion due to an overload of information [Mendling et al., 2007]. We have therefore not defined a candidate concrete syntax definition for the learning resource metadata. We do, however, demonstrate how a courseware *Topic* is associated with a learning *Resource* in section 5.6.4.

5.6 The Courseware Model

The CAVIAr Courseware Model defines the structure of courseware, from which its behaviour can be derived. There is a reference between the Courseware Model and the Learning Resource Model, and between the Courseware Model and the Learning Context Model. This allows for the Courseware Model to be evaluated with regard to the learning resources used and with regard to the learning context definition for the courseware.

In TEL there are two main specifications for describing courseware design, the ADL SCORM specification [ADL, 2004] and the IMS LD specification. Within the SCORM specification the IMS Simple Sequencing specification [IMS, 2003c] and content packaging specification [IMS, 2003a], in particular are used for courseware packaging and definition. Both SCORM and IMS LD are described in section 2.2.6.

In this section we will outline a language for defining courseware which is independent of (although inspired by) ADL SCORM and IMS LD. The courseware definition has been designed with validation in mind, concentrating on the courseware structure and sequencing. We defined a language independent of the main courseware specifications as this allows for minimal disruption to courseware validation tools, based on CAVIAr, during the maturing of the mainstream specifications, such as SCORM and IMS LD. Interoperability between CAVIAr and the mainstream TEL specifications will be addressed in section 8.8.

5.6.1 Defining an Abstract Syntax for Courseware Definition

The CAVIAr Courseware Model is defined in terms of its structure, in a hierarchical fashion. Learners move freely through courseware topics unless the course creator does not allow a certain movement. To allow courseware to be defined in this way we define an abstract syntax for specifying courseware structure. This abstract syntax is defined, in MOF, in figure 5.8.

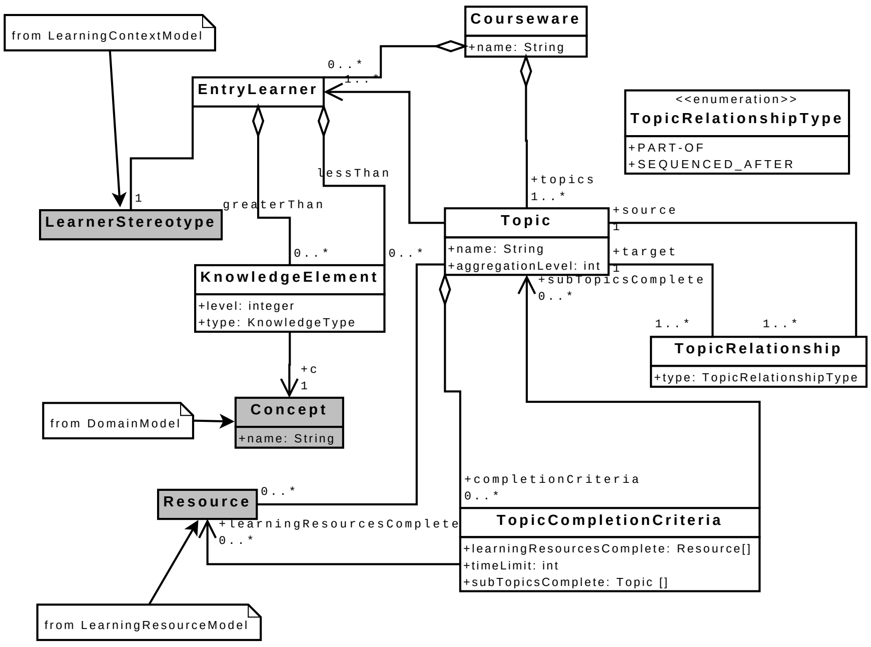


Figure 5.8: Abstract syntax for courseware defined using MOF

Courseware is defined using a courseware *name*. Courseware is principally comprised of *Topic* instances. A *Topic* has a *name* and an *aggregationLevel*. *Topics* are related to other *topics* via a *TopicRelationship*. A *TopicRelationship* has a *type*. The *type* is defined as a *TopicRelationshipType* enumeration. A *Topic* can be made up of other *topics* through the *TopicRelationshipType* - *PART_OF*. A *topic* which is “part of” another *topic* is a sub-topic of that *topic*. Explicit *topic* sequencing definitions are defined using the *TopicRelationshipType* - *SEQUENCED_AFTER*. The *SEQUENCED_AFTER* relationship specifies when one *topic* must be covered before another *Topic*. *Topics* can contain zero to many learning *Resources*. A topic *aggregationLevel* allows for the differentiation between different courseware granularity levels defined in section 2.2. A *topic* is generally set at a granularity level between the lesson granularity level and the course granularity level (defined in section 2.2).

An *EntryLearner* is a condition, associated with a *Topic* that must be true for a learner to enter the associated *topic*. The *EntryLearner* consists of one *LearnerStereotype*, defined in the Learning Context Model, and various *KnowledgeElements*, which are related to the *EntryLearner* through a *greaterThan* or *lessThan* relationship indicating if the learner must achieve greater than or less than the knowledge defined in the associated *KnowledgeElement*.

The *TopicCompletionCriteria* is contained within a *Topic*, it allows the course creator to express conditions for when the *Topic* is deemed to be complete. For example the course creator may only want the learner to complete two out of three of a topic's sub-topics. A *TopicCompletionCriteria* is expressed for each completion permutation. The *TopicCompletionCriteria* also allows for a simple time limit on a *Topic* instance.

The MOF defined abstract syntax is further constrained as follows:

- A *Topic* instance has an *aggregationLevel* between one and four.
- *Topics* referenced by a *TopicCompletionCriteria* instance must be related to the *Topic* that contains the *TopicCompletionCriteria* by a *TopicRelationship* of type *PART_OF*.
- *Resources* referenced by a *TopicCompletionCriteria* instance must be referenced by the *Topic* that contains the *TopicCompletionCriteria*.

These constraints are defined formally in OCL below in listing 5.3.

Listing 5.3: Courseware model abstract syntax constraints defined in OCL

```
context Topic
    inv: aggregationLevel_value: this.aggregationLevel > 0 and this.aggregationLevel < 5
    inv: isEmpty(self.completionCriteria->collect(subTopicsComplete) -
         self.relations->select(e|e.type=TopicRelationshipType::CONTAINS)->collect(target))
    inv isEmpty(self.completionCriteria->collect(subTopicsComplete) -
         self.resources)
```

5.6.2 Defining the Courseware Model Semantics

We consider courseware to be a Learning Activity LA, where LA is defined as the tuple $<SLA, LO, LP, SP, EP, n>$, SLA represents a set of embedded learning activities known as Sub-Learning Activities, LO is the set of Learning Objects associated with the Learning Activity, and LP is the set of learning paths. SP and EP represent the start points and end points respectively in the given Courseware Model. SP is optional to learning activities. A LA absent of a SP can be started at any sub-learning activity that does not have an incoming learning path. There may also be more than one SP for a LA. $SP, EP \neq LA$ and $SP, EP \neq LO$. The n variable represents the name for this learning activity.

- A learning path $lp \in LP$ is defined by the triple $<sla_1, sla_2, G>$, where $sla_1 \in SLA$, $sla_2 \in SLA$ are the source and target learning activities respectively. G is a set of boolean guard conditions on the learning path, defining when the learner may proceed from sla_1 to sla_2. G defaults to *true* when $G = \emptyset$.

- A gate condition $g \in G$ is defined as the tuple $<ls, KC>$, where ls is a learner stereotype defined in the Learning Context Model and KC is a set of knowledge conditions.

- A knowledge condition $kc \in KC$ is defined by the tuple $<com, KE>$, where KE defines a knowledge element, and the com is a comparator which defines whether the learner must be below or above the knowledge defined in the con.

- A learning object $lo \in LO$ is as defined in the LRM.

To complete the CAVIAr Courseware Model definition it is necessary to provide a mapping from the abstract syntax to the semantic domain. The courseware *Topic* is mapped to learning activity set LA. The containment relationship between a *Topic* and its sub-topics is mapped to the relationship between a learning activity la where $la \in LA$ and its set of sub learning activities SLA_{la}. There is an implicit learning path, lp, where $lp \in LP$ from each courseware *Topic* to every other *Topic* in the Courseware Model except for:

- Where there is a *TopicRelationship* of type *SEQUENCED_AFTER* between courseware *Topic* instances and the target topic or its container topics that have not been completed by the learner.

- Where the learner taking the courseware does not satisfy a topic's *EntryLearner* condition. *EntryLearner* conditions also hold for any of the topic's contained topics.

Instances of the *LO* MOF class are members of the LO set in the activity based syntax. Each learning activity has a set of start points SP and end points EP. The start point in a *Topic* is any topic which does not have an *EntryLearner* condition or *SEQUENCED_AFTER* condition (where that *Topic* is the source of the relationship) defined on it. The end point defined by default for a *Topic* is when all the topic's resources and contained topics have been delivered to the learner. A course creator can define a custom EP by defining a *Topic-CompletionCriteria* instance.

5.6.3 Courseware Model Interoperability

In order for courseware validation to be integrated into courseware construction tool support the Courseware Model must be interoperable with the state of the art in courseware specifications such as IMS LD, ADL SCORM and IEEE LOM. These courseware specifications are covered in detail in section 2.2.6. Here, we outline how the CAVIAr Courseware Model can be mapped to these courseware specifications.

The root of the abstract syntax is the *Courseware* class, this provides a container for the courseware specification. *Courseware* can be mapped to the *manifest* component in the ADL SCORM specification and to the *learning-design* component of the IMS LD specification. These are the root components of their respective specifications.

Courseware is broken down into logical units, known as *Topics*. A courseware *Topic* can be mapped to a *organization* and *item* in the ADL SCORM CP (Content and Packaging). In IMS LD the *Topic* can be mapped to the *play* and *act* constructs. The *Topic* class attribute *aggregationLevel* is used to allow the course creator to define different abstraction levels for courseware topics. For example an individual course creator may wish to make the distinction between courses, modules, lessons and learning resources in courseware, the topic *aggregationLevel* facilitates this, where a level is associated with a courseware abstraction level. The aggregation level can be used to distinguish between an "act " and "play" in IMS LD and an "organization" and "item" in SCORM.

The *TopicCompletionCriteria* can be mapped to the IMS Simple Sequencing "rollup rules", where the course creator can define when a learning activity is complete. As we have mentioned IMS simple sequencing can be defined at an item level in SCORM SN and can be defined on learning environments in the IMS LD specification. The relationship between a courseware *Topic* instance and a *Resource* instance can be mapped to the "using" relationship between an activity and resource in IMS LD, and between an *Item* and *Resource* in SCORM.

5.6.4 Defining a Candidate Concrete Syntax Definition

We define a concrete syntax for the Courseware Model through an example, as shown in figure 5.9. The figure shows a Courseware Model with three topics. The "SQL" *topic* and "JDBC" *topic* are sub-topics of the "Databases" *topic* as indicated by the *PART_OF TopicRelationship*, denoted as a solid line with a diamond at the target of the relationship.

In figure 5.9, the "JDBC" is sequenced after the "SQL" *topic*, as indicated by the broken arrow between the two *topics*. This broken arrow represents a *SEQUENCED_AFTER* type of *TopicRelationship*. The "SQL" *topic* contains a learning resource - "lo1".

The "JDBC" *topic* also has an entry learner condition - "EntryLearner1". The entry learner has a learner stereotype associated with it - "CS_Student". The concrete syntax also allows the course creator to define knowledge level conditions that must be satisfied for the learner to enter the topic. These are defined below the entry learner name, the first knowledge section defining knowledge levels of type *VERBAL_INFORMATION* and the second knowledge section defining *INTELLECTUAL_SKILLS* knowledge levels. The entry learner condition, in figure 5.9, states that the learner must have *INTELLECTUAL_SKILLS* knowledge in "Java" of greater than 0.6.

Figure 5.9 also demonstrates the use of the *TopicCompletionCriteria*, which is denoted as a small ellipse, with a numeric value in it. The numeric value represents the *TopicCompletionCriteria's timeLimit*. In the example, in figure 5.9, the notation also depicts that once "JDBC" is complete, "Topic1" is complete, or after thirty minutes "Databases" is complete.

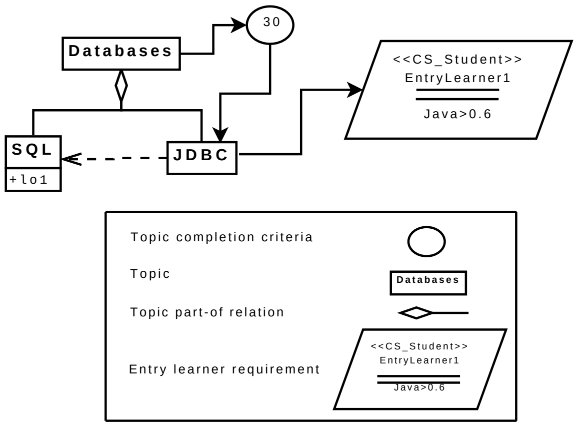


Figure 5.9: Example Courseware Model concrete syntax

5.7 Chapter Conclusion

In this chapter, we have defined the CAVIAr data models. To ensure each CAVIAr data model definition was as sound and as complete as possible, we grounded each CAVIAr data model definition in the courseware validation framework, defined in chapter 4. Each of the data models corresponds to one or more of the courseware construction concerns, as follows:

- The Learning Context Model captures the “courseware scope” concerns.
- The Courseware Model captures the “courseware structure and sequencing” concerns.
- The Learning Resource Model captures the “learning content in courseware” concerns.
- All the CAVIAr data models are used together to capture the “pedagogical strategy” concerns.

We defined each CAVIAr data model in terms of its abstract syntax using a MOF-defined metamodel, after this we defined a semantic domain for each metamodel using a graph-based notation. A semantic mapping was defined from the abstract syntax to the semantic domain. Interoperability is a key concern for the CAVIAr data models and as such each model has been mapped to a key TEL specification or standard.

Through the data models defined in this section it is possible to adequately describe courseware and courseware construction concerns in a standards neutral way. In chapter 6 we will outline how a constraints language can be used to ensure that the courseware construction concerns, defined primarily in the Learning Context Model, can be automatically validated. Chapter 6 also outlines how a constraint language allows the course creator to validate the courseware in other ways, including, ensuring the correct application of a particular instructional design.

In this chapter we have also outlined a candidate concrete syntax for the Domain Model, Learning Context Model and Courseware Model. The concrete syntax provides the course creator with a facade to the CAVIAr models hiding complexity. This is an important issue as the course creator is a non-technical end user, who requires a simplistic, intuitive user interface with which to define CAVIAr models. In chapter 9 we will evaluate the usability of this candidate concrete syntax.

Chapter 6

Courseware Validation

6.1 Introduction

In this chapter, we outline how CAVIAr allows for constructed courseware to be validated. Validation involves ensuring that courseware, represented by the CAVIAr Courseware Model, satisfies a variety of constraints, ranging from simple courseware structural constraints to complex constraints based on an instructional design theory.

We will use the OCL to define validation constraints. OCL is a constraint language for the metamodelling technical space designed to increase the semantic expressivity of UML and MOF models. OCL, UML and MOF are all defined by the OMG and as such designed to complement each other [OMG, 2003b]. OCL is a natural choice for defining constraints in CAVIAr as CAVIAr is defined in the metamodelling technical space using MOF. OCL is a language designed to be used by software engineers, not course creators. In section 6.2 we will outline our efforts to make defining courseware constraints in OCL more intuitive for the course creator. The first looks to define a set of common OCL functions for defining CAVIAr Validation Models that can be reused by the course creator. We also outline our efforts to develop a model-driven approach to defining constraints, where a model captures constraint data to be used to generate OCL. Subsequently, in section 6.3 our validation approach is described. We will identify three main categories of courseware validation; validation pre-requisites, courseware model validation and learning context validation. The subsequent sections describe each validation category of validation in detail. We summarise our findings in section 6.7.

6.2 Defining a Domain Specific Language for Constraints

As we have outlined in section 4.2, there are many actors involved in courseware construction. The instructional designer programmer defines the definition of the CAVIAr Validation Model. The instructional designer programmer is an expert in the application of instructional design theory to a specific courseware, defined for a specific learning context.

Converting instructional constraints into OCL is not a trivial task. We aim to support the instructional designer programmer in this task by defining a Domain Specific Language (DSL) to define constraints. To do this we have investigated two approaches for this, defining OCL helper operations and creating an intuitive DSML for OCL. The helper operations abstract commonly-used and complex constraint patterns in courseware validation, and define them as an OCL operation. This creates a more high level and intuitive constraint language for the instructional designer programmer to define the courseware validation criteria with. We describe this approach in the next section. The second approach defines a DSML for instructional design constraints. The DSML represents courseware validation constraints using an intuitive graphical notation. The instructional designer programmer uses the DSML to define validation criteria. The DSML is then used to automatically generate the CAVIAr Validation Model in the form of OCL invariants. This approach is outlined in section 6.2.2.

The two approaches we outline extend research by Wahler et al. This research attempts to simplify OCL construction by defining an extensible library of generic OCL patterns [Wahler et al., 2006].

6.2.1 OCL Helper Operations

Our first approach allows for the extension of OCL with a domain specific vocabulary that queries CAVIAr models for specific elements. This is done by outlining a variety of operations or functions that can be used by the instructional designer programmer when defining a CAVIAr Validation Model in OCL. It should be noted that this mechanism of raising the course creator's level of abstraction is highly customisable and extendable; an instructional designer programmer may even define their own operations. Operations defined by other instructional designer programmers can also be used when defining constraints, in much the same way as programming libraries can be imported to raise the level of abstraction an application programmer develops at.

A helper operation is defined using the OCL *def* construct. The operation is given a name, parameters and a return type. The operation is basically an OCL query from the defined model *context*. The query must return a datatype as specified by the helper operation return type.

An example of where a helper modelling operation might be used is where a constraint must be defined on a CAVIAr class that is used to capture relationship semantics between two instances of the same CAVIAr metamodel class. An example of this type of relationship can be found at the *Topic* class in the Courseware Model's metamodel and the *Concept* class from the Domain Model's metamodel. In order for the semantics of the association between the two modelling elements to be captured, the association itself is represented as a separate class in the metamodel. In figure 6.1, we illustrate one of the aforementioned relationship classes, where topics are related to each other using an instance of the *TopicRelationship* class, which can relate a *Topic* instance to another *Topic* instance. The semantics of the *TopicRelationship* are defined in the *type* attribute of the *TopicRelationship* class.

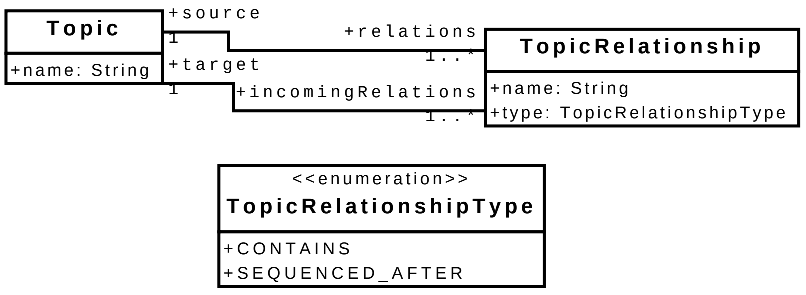


Figure 6.1: Extract of the CAVIAr Courseware Model's metamodel

In order to define constraints on CAVIAr model definitions in OCL the *TopicRelationship* class must be considered to evaluate the *type* relationships between topics. To illustrate this point we have defined an OCL constraint in listing 6.1 stating if a topic contains other topics, the contained topics must reference more learning resources than that of the containing topic. The invariant must be defined by traversing the *TopicRelationship* class where the TopicRelationship's *type* attribute is *CONTAINS*.

Listing 6.1: Invariant definition over two related topics using the TopicRelationship class

```
context Topic
    inv more_LOs_in_contained_topics : self.relations
        ->select(type = TopicRelationshipType::CONTAINS)
        ->forall(resources->size() >= self.resources->size())
```

The constraint defined in listing 6.1, demonstrates where the semantics of the *TopicRelationship* class are important. Although it is necessary to evaluate the relationship class, it is not a very intuitive way of constructing invariants. The main cause for confusion is that relationships are represented in the metamodel as classes. While in a CAVIAr model, which is defined and used by the course creator, a relationship is typically represented as a line, with different types of lines indicating the semantics of the relationship. This is demonstrated in figure 6.2 where the notation of a line with a diamond at one end is used to denote the type is *TopicRelationshipType::CONTAINS*.

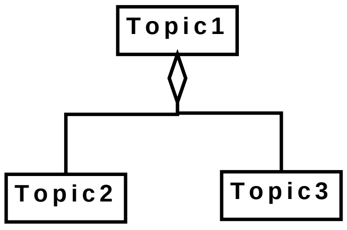


Figure 6.2: CAVIAr Courseware Model extract

Listing 6.2: CONTAINS derived attribute definition

```
context Topic
    def: getContainedTopics(): set(Topic)=self.relations
        ->select(type=TopicRelationshipType::CONTAINS)
        ->collect(target)
```

In order to make constraints based on these relationships more intuitive, we define an OCL operation that encapsulates the OCL that is needed to traverse such a relationship in a metamodel. In listing 6.2 we illustrate the definition of the *getContainedTopics()* operation in the context of the *Topic* class. This operation provides OCL so that the course creator is not required to define the OCL to navigate the *TopicRelationship* class. The direct relationship is more intuitive for the instructional designer programmer.

This operation definition allows for the more intuitive definition of the invariant defined in listing 6.1 on the Courseware Model's metamodel. In listing 6.3 we use the new operation to define the original invariant in listing 6.1.

Listing 6.3: Invariant definition using the CONTAINS derived attribute

```
context Topic
    inv More_LOs_in_contained_topics : self->getContainedTopics()->forAll(resources->size()
        >= self.resources->size())
```

Although we have concentrated on *Topic* here, the principles presented can be applied to any relationship in the metamodel where the semantics of a relationship are captured in a metamodel class. Indeed the general principles can be applied to anywhere in the CAVIAr model and are of particular use for streamlining the use of commonly used OCL.

In table 6.1 and 6.2, we outline OCL helper operations that we have defined. The tables are organised by the model element they work on. Table 6.1 outlines operations defined for the Learning Context Model, while table 6.2 outlines operations defined for the Courseware Model. The decision on what operations to define were empirically determined and validated through a related project [Janjua, 2008]. Our approach raises the level of abstraction that the instructional designer must define the CAVIAr Validation Model at. This allows him or her to concentrate on the business of defining instructional constraints rather than defining complex model navigation using OCL. Table 6.1 and 6.2 do not represent a complete list of all possible reusable operations in CAVIAr, the instructional designer programmer may add to this by defining their own OCL operations.

Table 6.1: Derived CAVIAr operations for Learning Context Modelling constructs

Operation Name	Operation Context	Description
containedGoals()	Goal	Operation gets all the goals contained in the context goal.
altGoal()	Goal	This operation returns any alternative goals for the context goal.
narrower()	Concept	This operation returns Concepts that are related to the context concept via ConceptRelationship of type NARROWER - direction source to target.
broader()	Concept	This operation returns Concepts that are related to the context concept via ConceptRelationship of type NARROWER - direction target to source.
siblings()	Concept	This operation returns a Concept's sibling concepts (i.e. concepts with the same parent in the semantic scope taxonomy).
prerequisite(level:Integer,includeVB:Boolean)	Concept	This operation returns Concepts that are related to the context topic via ConceptRelationship of type PRE_REQUISITE. Concepts that are returned must be related via a knowledge element above the level stated in the *level* parameter. Verbal information pre-requisites are included when *includeVB* is true.
getPresumedLearnerConcepts()	Learner	This operation returns a set of concepts the learner is assumed to know when starting the course.

Table 6.2: Derived CAVIAr operations for Courseware Model constructs

Operation Name	Operation Context	Description
containedTopics()	Topic	This operation returns topics contained within the context Topic
sequencedAfterTopics()	Topic	This operation returns Topics that are related to the context topic via TopicRelationship of type SEQUENCED_AFTER.
containerTopic()	Topic	This operation returns Topics that are related to the context topic via TopicRelationship of type CONTAIN.
getTopicConcepts()	Topic	This operation will get the intersection of the concepts covered by LOs in the context topic.
getAllTopicConcepts()	Topic	This operation will get the union of all the concepts covered by all LOs in the context topic.
getConceptualPreReqTopics(level:Integer,includeVB:Boolean)	Topic	This operation returns a set of topics that cover concepts deemed pre-requisite concept to the concepts covered in this topic. Topics that are returned must be have a conceptual relation with a knowledge element above the level stated in the *level* parameter. Verbal information pre-requisites are included when *includeVB* is true.
getLOPreReqTopics()	Topic	This operation returns a set of topics that contain LOs which are pre-requisite to the LOs covered in this topic.
getCoursewareTopics()	Courseware	This operation returns a set of all topics in a given courseware.
getCoursewareConcepts()	Courseware	This operation returns a set of concepts covered in a given courseware.
getCoursewareResources()	Courseware	This operation returns all resources in the context courseware.
getCommonPresumedKnowledge(level:Integer,includeVB:Boolean)	Courseware	This operation gets the union of all the presumed knowledge for all the learners who it is anticipated will use this courseware. To be included the knowledge level must be above the level set by *level*, verbal information is only included if *includeVB* is true.

6.2.2 Towards a Model-Driven OCL Generation

We have raised the level of abstraction that the course creator creates a CAVIAr Validation Model at using a model-driven approach. This was done through a related project where we developed tool support for the course creator allowing him or her to define CAVIAr Validation Model constraints using an intuitive DSML [Janjua, 2008]. The tool developed is based around CAVIAr Validation Model constraint patterns. A CAVIAr Validation Model constraint pattern is an abstract definition of some instructional constraint. Constraint patterns are determined for each of the CAVIAr Courseware Model constructs. Patterns are relatively simple, frequently used validation constraints defined on a particular courseware construct. A constraint pattern is instantiated by the course creator using instance specific data about the constraint.

We identified CAVIAr Validation Model constraint patterns for *Topic* and *Courseware* constructs from the CAVIAr Courseware Model by analysing the type of constraints defined for validation and identifying frequently used constraint structures. These were as follows:

- Topic Patterns
  - Timing - Restriction on the length of time a topic can take.
  - Learning Resource - Checks that a learning resource type is present or not present in a given courseware topic.
  - Adaptivity - Verifies that a particular type of learner is considered for a given courseware topic.
  - Structural - Checks the number of topics contained by a given courseware topic and the aggregation level of the contained topics.
- Courseware Patterns
  - Timing - Restriction on the length of time courseware can take.
  - Structural - Checks the structure of the courseware in terms of the topics it contains and how topics are sequenced in the courseware

The data needed to define each of these constraint patterns was elicited and used to define a constraint pattern metamodel using ECore. An example of such an metamodel can be found in figure 6.3, where the topic structural constraint metamodel is defined using MOF. This constraint shows that there are two main pieces of data needed to define a topic structural constraint, these being the number of contained topics within a given topic and the aggregation level of those contained topics. The model allows for the explicit representation of this data.

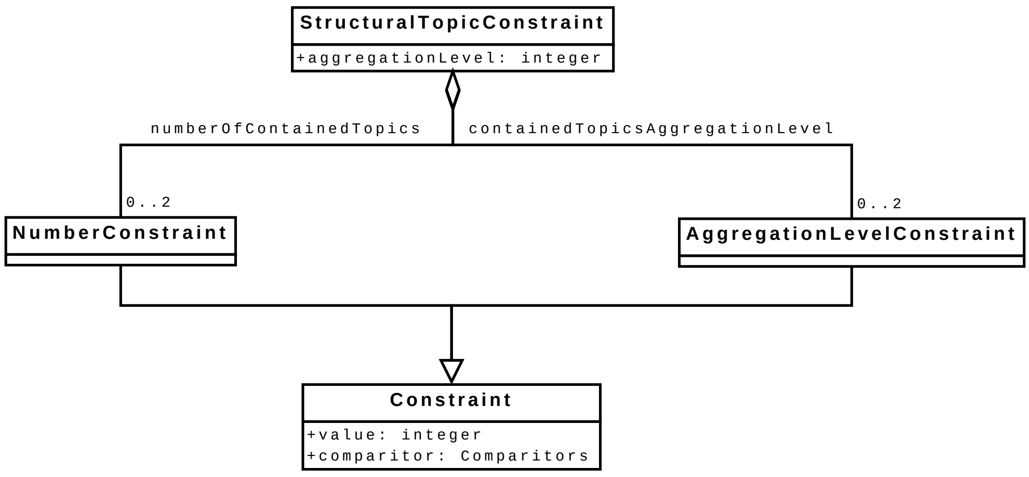

StructuralTopicConstraint
+aggregationLevel: integer
numberOfContainedTopics
containedTopicsAggregationLevel
0..2
0..2
NumberConstraint
AggregationLevelConstraint
Constraint
+value: integer
+comparitor: Comparitors

Figure 6.3: MOF model used to capture data about the topic structural constraints

Our tool support allows the course creator to define CAVIAr Validation Model constraints using an intuitive model-based concrete syntax for each constraint pattern. In figure 6.4 we outline an example of our intuitive model based concrete syntax. In this case the course creator has defined a timing constraint on courseware *Topics*. The grey box defines the set of *Topics* that this constraint is defined on. In this case the constraint is defined on courseware *topics* associated with the the “Software Quality” Domain Model *concept* and have an *aggregation level* of one. The blue box connected to the *Topic* represents a timing constraint. In this case the timing constraint defines a maximum time 30 minutes and a minimum time of 20 minutes. As this timing constraint is associated with the *Topic* set definition, all topics in the set defined by the grey box must comply with this timing constraint. The tool support was developed using the Eclipse Graphical Modelling Framework (GMF).

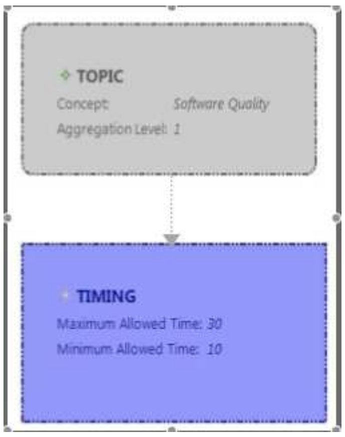


Figure 6.4: Model-based definition of a courseware topic timing constraint

By defining a CAVIAr Validation Model using our GMF-based tool support, the course creator creates an instance of a constraint pattern metamodel - a constraint pattern model. Constraint pattern models are formalised into OCL using a model-to-text (M2T) generator, such as JET (JET is covered in section 8.4.3). The OCL constraint is then integrated into a CAVIAr Validation Model. As this approach is extended to cover more constraint patterns the course creator will be able to use this intuitive model-driven approach to define a complete CAVIAr Validation Model.

6.3 Validation Approach

The CAVIAr data models, which define the courseware and courseware construction concerns, are defined using MOF in chapter 5. The abstract syntax (metamodel) of a CAVIAr data model is defined using MOF. MOF metamodels can be constrained using OCL, as described in section 2.4. The course creator can define constraints, in OCL, on the Courseware Model's metamodel that must be true for a Courseware Model to be valid. The course creator can use data defined in the Learning Context Model to define these constraints. In figure 6.5, we illustrate this diagrammatically.

We have identified three principle categories of validation, these are [Melia and Pahl, 2009]:

- Validation Pre-requisites - This type of validation does not check for instructional

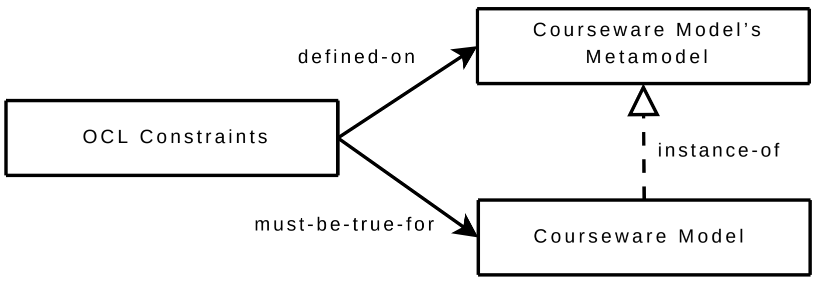


Figure 6.5: Using OCL to define constraints on a metamodel that must be true for models that conform to that metamodel

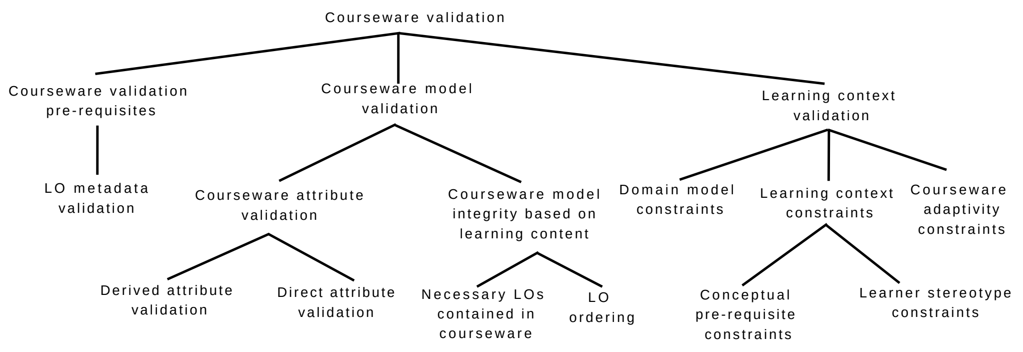
features in the CAVIAr models, but checks that a minimum amount of data is available in the models to allow for validation. Validation pre-requisites do not address any of the courseware construction concerns covered in section 4.3. The validation pre-requisites allow the course creator to have greater confidence in validation, as it guarantees that all data needed for validation is available.

- Courseware Model Validation - validation based on the courseware design represented in the CAVIAr Courseware Model and it related Learning Resource Model. This type of validation examines the LOs used in the courseware, and how the course LOs are related to each other and grouped together into topics. Courseware Model validation is concerned with the following courseware construction concerns:
  - Structure and Sequencing - Validates the courseware structural integrity.
  - Learning Content in Courseware - Checks what content is in courseware, where it is in courseware and how it is used.
  - Pedagogical Strategy Used - Used to validate the pedagogical strategy such as the implementation of an instructional design theory.
- Learning Context Validation - This type of validation makes extensive use of the CAVIAr learning context definition (see section 5.4), by comparing the Courseware Model to the Learning Context Model. Learning context validation is concerned with the following courseware construction concerns:

- Course Scope - Ensures that all learning goals can be reached in courseware and that the courseware considers the learner's assumed initial knowledge.
- Structure and Sequencing - Concerned with conceptual sequencing defined in the Learning Context Model.

Each of the constraint categories can be further classified as outlined in figure 6.6. We will examine each category in detail as follows:

- Section 6.4 describes how validation pre-requisites can be defined using OCL.
- Section 6.5 examines the courseware model validation category and outlines how to define these types of constraints using OCL.
- Section 6.6 looks in detail at courseware validation that is based on the Learning Context Model and how these can be defined in OCL.



Courseware validation
Courseware validation pre-requisites
Courseware model validation
Learning context validation
LO metadata validation
Courseware attribute validation
Courseware model integrity based on learning content
Domain model constraints
Learning context constraints
Courseware adaptivity constraints
Derived attribute validation
Direct attribute validation
Necessary LOs contained in courseware
LO ordering
Conceptual pre-requisite constraints
Learner stereotype constraints

Figure 6.6: Classification of CAVIAr validation constraints

6.4 Courseware Validation Pre-requisites

The instructional designer programmer sets validation pre-requisites as constraints on the CAVIAr model that must be true for validation to take place. The main reason behind validation pre-requisites is to ensure that the data required for validation is present in the CAVIAr models.

Much of CAVIAr validation requires specific LO metadata in order to accurately determine the validity of courseware. LOs used in the courseware must be annotated with certain metadata for the course creator to be confident in validation. The course creator can specify what metadata must be present in the CAVIAr data models in order for validation to go ahead.

An example of this type of rule is outlined in listing 6.4 where the course creator outlines that *Metadata* classes in a Courseware Model must be fully annotated with educational metadata (i.e. an educational metadata instance exists and all educational metadata types do not equal null).

Listing 6.4: Courseware element integrity rule ensuring all LOs have educational metadata

```
Context Metadata
    inv metadata_must_have_educational: !self.educational.isOclUndefined() and
        self.educational.interactionType <> null and
        self.educational.interactivityLevel <> null and
        self.educational.semanticDensity <> null
```

6.5 Courseware Model Validation

In CAVIAr, the Courseware Model defines the structure of courseware using courseware topics. A *Topic* sequencing strategy can be specified by stating that one topic must be sequenced after another courseware *Topic* using the *TopicRelationship - SEQUENCED_AFTER*. Learning resources are conceptually contained in topics. Courseware personalisation is determined by specifying guard conditions on courseware topics, known as *EntryLearner* conditions. For a learner to enter a given topic the *EntryLearner* condition must be true for that learner.

Courseware model validation looks to validate the Courseware Model in isolation of the learning context. Validation based solely on the Courseware Model lends itself to two types of validation:

- Courseware Attribute Validation - This type of validation validates a courseware attribute against an externally defined value.

- Courseware Model Integrity Based on Courseware Learning Content - Validates Courseware Model, ensuring it is structured correctly for the learning content it contains.

6.5.1 Courseware Attribute Validation

This is the simplest type of validation the course creator can define. It involves comparing an attribute of the Courseware Model with an external value, or deriving a value from a defined set of courseware attributes and comparing that to some external value. The external value is an alphanumeric value. The comparison tests the relation between that external value and the one from the Courseware Model using a relational operator.

Here, we will outline a variety of simple instructional constraint rules that the course creator can use to validate the courseware. Constraints are defined using some simple external requirement that is compared with an attribute from the LO's metadata or compared with a derived value based on a grouping of LOs such as that found in a courseware *Topic*.

6.5.1.1 Defining Validation Rules

To define this type of rule the course creator must identify the CAVIAr context class that the OCL invariant will be defined on. The course creator then defines the instructional constraint by comparing a courseware value with an external comparison value. To define the courseware value the course creator either specifies an attribute of the context class or defines a collection using the context class's associations and derives a value from the collection defined. To derive a courseware value from a collection, the course creator can use one of the OCL collection operations (see section 2.4.1.8).

If the courseware value and the external value are numeric, a comparison is done using a relational operator. If on the other hand the values are strings only the equals and not equals relational operators are valid.

6.5.1.2 Validation of Individual Courseware Model Element Attributes

Defining constraints on individual Courseware Model elements is done by specifying constraints on the courseware and/or Learning Resource Model, constraining the attributes allowed in these models.

To demonstrate the validation of individual elements of constructed courseware we define an example constraint. In this example we constrain what LOs can be used in the courseware by specifying a maximum duration time for LOs used in courseware. This constraint is defined in listing 6.5. This type of rule might be used in an environment where the learner's time is an expensive resource.

In our example case, we wish to limit the duration of each LO in the Courseware Model to thirty minutes. In order to evaluate this the duration of each LO must be evaluated ensuring its duration time is below the maximum specified time. The duration of each LO can be found in the LO's technical metadata. In listing 6.5, the rule's context is the *Technical* class in the Learning Resource Model. The invariant specifies that instances of the *Technical* metamodel class must have a *duration* (attribute of *Technical* class) of less than thirty.

Listing 6.5: OCL rule which specifies LOs in CAVIAr cannot be longer than thirty minutes in duration

```
context Technical
    inv LO_violate_max_duration_time: duration < 30
```

6.5.1.3 Validation of Derived Courseware Attributes

Data can also be derived from the Courseware Model. This data can be used to validate aspects of courseware. To do this, the course creator queries a set of CAVIAr Courseware Model elements and derives some data from it. Using OCL, the course creator can specify constraints on this derived data from the Courseware Model. To demonstrate this we outline an example that uses OCL to derive the duration of courseware as a whole and compares this against some maximum courseware time value. In this example, the LO duration values are used to calculate the duration of the courseware as a whole. This involves specifying how to evaluate the time of courseware topics and then specifying a courseware invariant that calculates the sum of all topic times in the courseware. This value is then compared to the maximum time value that a courseware can take and ensures courseware does not exceed it.

In listing 6.6, the course creator firstly defines the topic operation *getTopicTime()* to calculate the duration of courseware *Topics*. The course creator must define this as it is not one of the helper operations defined in table 6.2. This demonstrates how the course creator can extend the operations we have defined and define their own helper operation.

The *getTopicTime()* operation is defined with a context of *Topic* and recursively traverses contained topics. A topic's time is calculated as the sum of the LOs associated with that *Topic* and the duration of its contained Topics. The invariant *max_courseware_time_exceeded* is then defined in listing 6.6, which uses the *getTopicTime()* operation to get the duration of each of the topics associated with the *Courseware* instance. The sum of the duration attribute of each topic is then compared with the maximum courseware duration, which in this case is one thousand (minutes).

Listing 6.6: OCL constraint that evaluates the courseware time from its contained LOs and specifies a maximum courseware time of 1000 minutes

```
Context Topic
    def: getTopicTime(): Integer =
        self.resources->select(oclIsOfType(LO))
        ->iterate(i; res:Integer=0| res = res+i.oclAsType(LO).metadata.technical.duration)
        +self.containedTopics()->iterate(j;a:Integer=0|a= a+j.getTopicTime())
Context Courseware
    inv max_courseware_time_exceeded :
        self.topics->iterate(i;res:Integer=0|res=res+i.getTopicTime()) < 1000
```

This type of courseware validation rule can also be used to validate simple elements of the courseware structure, where the courseware structural feature can be resolved to a simple data type. An example of such a validation constraint might be enforcing a minimum amount of LOs contained by each courseware topic. In listing 6.7, we specify that each topic in the courseware must have at least ten learning objects contained in it.

Listing 6.7: OCL invariant specifying a minimum number of LOs in a courseware topic

```
context Topic
    inv min_LOs_in_topic: self.resources
       ->select(oclIsOfType(LO))->size() > 10
```

6.5.2 Courseware Integrity based on Courseware's Learning Content

Here, we check that the courseware is structured correctly for the learning content it contains. An example of such validation criterium is to ensure that all courseware learning resources that are referenced by a LO contained in the courseware are also covered by some topic in the courseware. More complex integrity checks, check the sequencing of learning resources that reference each other, ensuring that the delivery sequence in the courseware corresponds to how the learning resources are reference each other.

6.5.2.1 Necessary Learning Objects contained in Courseware

In listing 6.8 we have outlined an invariant that is deemed valid if all resources referenced by resources in the courseware are also contained somewhere in the courseware. In defining this rule the course creator defines how to get a set of all the resources that are needed in the courseware (i.e. those referenced by a LO in the courseware). This is achieved by defining the local variable *neededResources*. This variable is defined by querying each of the LOs in the courseware to see if it references another *Resource*, if it does the referenced *Resource* is added to the result set. When all resources have been evaluated the result set is returned. The invariant, *all_referenced_LOs_in_courseware* is valid if by subtracting the LOs covered in the courseware results in an empty set.

Listing 6.8: OCL ensuring that referenced LOs are in courseware

```
Context Courseware
    inv      all_referenced_LOs_in_courseware:
    let      neededResources:Set(Resource)=
        getCoursewareResources()−>select(oclIsTypeOf(LO))
        −>iterate(r:Resource; los:Set(LO)=Set{}|los−>union(r.oclAsType(LO)))
        −>iterate(y:LO; a:Set(Relation)=Set{}| a−>union(y.metadata.relations))
        −>iterate(x:Relation; res:Set(Resource)=Set{}| res−>union(x.referencedResource))
    in
         self.neededResources − self.getCoursewareResources()=Set{}
```

6.5.2.2 Relationships between Learning Objects are Respected in Courseware

The constraint rule in listing 6.9 can be further refined by checking the semantics of the relationship between learning resources in the courseware. For example, should the relationship be of type *RelationKind::BASED_ON*, such that LO_b is based on LO_a, the instructional designer programmer can ensure that LO_a is sequenced first. This is done by defining an OCL constraint that specifies there must be a *SEQUENCED_AFTER* topic relationship between the topic that contains LO_a and LO_b, stating that the topic containing LO_a is sequenced first.

Listing 6.9: OCL ensuring that referenced LOs are in courseware

```
Context Topic
    inv LO_based_on_reference_respected: resources->select(oclIsTypeOf(LO))
        ->iterate(r:Resource; los:Set(LO)=Set{}| los->union(r.oclAsType(LO)))
        ->iterate(y:LO; a:Set(Resource)=Set{}| a
            ->union(y.metadata.relations->select(kind=RelationKind::BASED_ON))))
        -
         self.sequencedAfterTopics()
        ->iterate(t:Topic; sLos:Set(Resource)=Set{}|sLos->union(t.resources))
         =Set{}
```

To define this type of rule, the course creator builds up two sets describing the two aspects of the courseware, which are to be compared. In this case, the first set defines the learning resources that the context topic's learning resources are based on and the second set defines all the learning resources of the topics sequenced after the context topic. A set operation is then used to compare the sets. In this case we specify that the difference between the learning resources sequenced after the context topic and the learning resources that the context topic learning resources are based on must result in an empty set.

This type of rule can also be used to ensure that specific types of LOs are sequenced in a particular way when teaching a give concept. This allows for the validation of intra-conceptual sequencing patterns, such as planning sheets defined by Gagné et al., which is outlined in detail in section 2.2.2.

6.6 Learning Context Validation

The Learning Context Model, covered in detail in section 5.4, defines courseware learner stereotypes and also conceptual sequencing constraints, both of which are defined in terms of the Domain Model. The learner stereotypes are defined in terms of their conceptual learning goals and presumed knowledge. In this section, we examine the relationship between the learning context definition and the courseware developed. Our aim here is to define constraints on the Courseware Model in terms of the learning context definition. This ensures that courseware adheres to the course scope and conceptual sequencing requirements stated in the Learning Context Model.

In order to validate the courseware against the course requirements defined in the Learning Context Model, the learning context must be defined in the context of the Courseware Model. We will outline how a relationship between the Learning Context Model and the Courseware Model can be established allowing the course creator to define an interpretation of the learning context. This is done using OCL.

Here, we define the three types of instructional constraints that are defined using the learning context:

- Domain Model Constraints - Instructional constraints using the Domain Model only.
- Learning Context Constraints - Instructional constraints using the overall learning context including the Domain Model and the learner stereotype information in CAVIAr.
- Courseware adaptivity constraints - These constraints ensure that personalisation used in the courseware is instructionally valid.

6.6.1 Domain Model Constraints

The Courseware Model is associated with a Domain Model through LO metadata. Each LO can be classified to one or more Domain Model concepts. This association can be used to examine the Courseware Model structure in the context of the Domain Model structure. Validation constraints can then be defined based on a comparison between the Domain Model and the Courseware Model. Here, we will look at how OCL can be used to compare a Courseware Model with its related Domain Model to determine the courseware's validity.

Conceptual relationships in a Domain Model define how two concepts are related to each other. These relationships can be used to derive instructional design rules that can then be validated against constructed courseware. For example the Domain Model's *NARROWER* concept relationship could be used to define an implicit sequencing constraint between courseware topics that have LOs, which reference the relationship's *source* concept and LOs, which reference the relationship's *target* concept.

The narrower conceptual relationship is used in the instructional constraint defined in listing 6.10. The constraint specifies that topics covering more specialised (narrower) concepts must be sequenced after topics covering more abstract (broader) concepts.

Listing 6.10: OCL constraint using conceptual relationship semantics to define an instructional constraint where all topics covering broader concepts are sequenced before those covering more specialised concepts

```
Context Topic
    inv:      self.getTopicConcepts()
             ->iterate(x:Concept; a:Set(Concept)=Set{}| a->union(x.broader()))
        - self.sequencedAfterTopics()
             ->iterate(y:Topic; b:Set(Concept)=Set{}| b->union(y.getTopicConcepts()))
        = Set{} --empty set
```

To formulate this type of rule the course creator creates two sets to compare, one set based on a Domain Model traversal, using the inverse of the *NARROWER* concept relationship (set A), and one based on a Courseware Model traversal using the *SEQUENCED_AFTER* topic relationship (set B). The two sets are compared using the difference OCL collection operator which should result in an empty set, $A - B = \emptyset$. This verifies that all concepts broader than the concept(s) covered by the context topic are covered before the context topics. In listing 6.10, these two sets are defined and compared using an invariant which is valid if the difference operation results in an empty set.

6.6.2 Learner Context Constraints

Here we examine the two types of constraints that can be defined using the Learning Context Model, these are:

- Conceptual Pre-requisite Constraints - Conceptual sequencing constraints on the Domain Model. Here we assume one common Domain Model for all LO conceptual annotations in courseware.

- Learner Stereotype Constraints - The course creator can use a learner stereotype to define learning goals and the presumed knowledge for a learner grouping. This information can be used to ensure that the needs of a learner stereotype grouping are met by courseware.

6.6.2.1 Conceptual Pre-requisite Constraints

A seemingly obvious choice for checking pre-requisite constraints on courseware topics would be to use the OCL pre- and post-condition constructs (see section 2.4.1.5). After investigating the use of OCL pre- and post-conditions we found that the semantics of these OCL constructs were not appropriate for checking conceptual pre-requisite constraints. The pre-conditions and post-conditions in OCL are designed to check for a particular state prior to and after executing an operation at runtime. To explain this we consider the OCL pre-conditions and post-conditions in terms of the OMG modelling layers (see section 2.3.1), where runtime is defined as the M0 modelling layer. M0 is the only layer an operation pre- and post-condition can be checked as operation execution is a runtime event. Pre- and post-conditions are therefore defined at the M1 modelling layer. Courseware validation checks the courseware structure at the M1 modelling layer, and is defined at the M2 modelling layer, therefore OCL pre-conditions and post-conditions cannot be used to define conceptual pre-requisite constraints specified in the CAVIAr Learning Context Model. To validate the M1 modelling layer OCL constraints are defined at the M2 modelling layer - the CAVIAr metamodel. OCL invariants use the data from the Learning Context Model to define what is a valid and invalid courseware structure. When courseware is defined, an OCL checker can be used to check that the constraints imposed by the Learning Context Model on the Courseware Model are satisfied by defining OCL invariants based around the sequencing of topics and the conceptual pre-requisite relationship defined as part of the Learning Context Model. The use of the “invariant” type of OCL constraint also limits the OCL diversity used in formulating the CAVIAr Validation Model. The CAVIAr Validation Model is therefore constructed in a way where all constraints are defined using a uniform mechanism. This limits the learning curve the instructional designer programmer is faced with when defining a Validation Model and also limits the diversity that CAVIAr tool support would have to handle.

The first invariant we define builds on the definition of the *getTopicConcepts()* operation of the *Topic* class, defined in table 6.2. This invariant checks that if there is a pre-requisite relationship between two concepts where concept c_{pre} is the pre-requisite of concept c, then c_{pre} will be covered in the courseware before concept c. We have defined this constraint as an OCL invariant in listing 6.11.

Listing 6.11: OCL to ensure that pre-requisite concepts are always sequenced before the topic which requires it

```
context Topic
    inv conceptual_prerequisite_rule: self.getAllTopicConcepts()
           ->iterate(x:Concept; a:Set(Concept)=Set{}| a->union(x.prerequisite(0.5, true)))
        - self.sequencedAfterTopics()
           ->iterate(y:Topic; b:Set(Concept)=Set{}| b->union(y.concepts))
         = Set{}
```

In this constraint (listing 6.11) the *Topic* class is defined as the invariant context, as each of the topics in the courseware must be checked against this constraint. The course creator defines two sets, the first containing all the pre-requisite concepts of the concepts covered by the context topic, set P, and the second set containing the concepts that will definitely be covered prior to the learner getting to the context topic, set C. The pre-requisite constraints set is defined as any conceptual pre-requiste relationship with a knowledge type *VERBAL_INFORMATION* or *INTELLECTUAL_SKILLS* and a knowledge level of greater than 0.5 (see table 6.1 for details on the *prerequisite* OCL operation). The second set is constructed by getting the concepts covered by LOs at topics that are related to the context topic through the *SEQUENCED_AFTER* relationship. The *SEQUENCED_AFTER* relationship guarantees that the target topic will be sequenced after the source topic. The difference between these two sets is then sought using the OCL difference operator, which must result in an empty set (i.e. there are no concepts which are pre-requisite concepts and not covered by topics sequenced before the topic in question, $P - C = \emptyset$). The rule in listing 6.11 illustrates how a basic conceptual pre-requisite constraint can be defined based on the Learning Context Model. This constraint could be extended to consider, for example, transitive sequencing relationships in the courseware or Learning Context Model.

The constraint in listing 6.11 allows us to check the sequencing of concepts covered in courseware ensuring that pre-requisite knowledge is sequenced first. This rule does not take into account a learner's knowledge prior to taking the courseware (defined here as set L). For example if the set resulting from the difference operation resulted in one concept, c_1, where $c_1 \in P$ i.e. the concept is a pre-requisite concept, but this concept is presumed knowledge of a particular learner stereotype taking the courseware. This courseware should still be deemed valid for that learner stereotype as the concept, although not covered prior to the topic that needs it, is knowledge the learner already has.

In listing 6.12 a generalistic approach to dealing with learner pre-requisite knowledge is taken, as follows: $(P - C) - L = \emptyset$. The conceptual pre-requisite constraint violation is sought first as in listing 6.11, then any violating concepts are compared against the common assumed knowledge for all learner stereotypes to take this courseware. Only when there are still outstanding concepts will the invariant *conceptual_prerequisite_rule* in listing 6.12 be flagged as invalid.

Listing 6.12: OCL ensuring conceptual pre-requisites are sequenced before topics that require them and also that assumed learner knowledge is acknowledged

```
Context CAVIAr
    inv conceptual_prerequisite_rule: self.courseware.getCoursewareTopics()
        ->forAll(
            (getTopicConcepts()->iterate(x:Concept; a:Set(Concept)=Set{}|a->union(x.prerequisite(0.5, true)))
            -sequencedAfterTopics()->iterate(y:Topic; b:Set(Concept)=Set{}| b->union(y.concepts))
            -self.learningContext.ls->
             iterate(x:LearnerStereotype;
                 a:Set(Concept)=self.learningContext.ls->first().constraints
                ->select(oclIsOfType(PresumedKnowledge))
                ->iterate(i;res:Set(Concept)=Set{}|res->including(i.competency.c))  |
            x.constraints->select(oclIsOfType(PresumedKnowledge))
            ->iterate(j;res2:Set(Concept)=Set{}|res2->including(j.competency.c))
            ->intersection(a) = Set{}
        )
```

To construct the invariant in listing 6.12 the course creator must change the context from *Topic* to *CAVIAr*, as the learner stereotypes cannot be navigated to from a *Topic* context. The constraint is defined to traverse each *Topic* and query each of them for pre-requisite

Concepts that are not covered by topics sequenced before it. If pre-requisite concepts are found that are not covered by the topics sequenced before the current topic, the invariant compares these concepts with the presumed learner conceptual knowledge, derived as the set of concepts that all learner stereotypes are assumed to know (intersection of all presumed knowledge concepts). If any pre-requisite concepts are not covered in a previous topic and not in the set of presumed learner knowledge the invariant is not satisfied.

6.6.2.2 Validating Learner Stereotype Course Goals

The CAVIAr learner stereotype construct is based on two forms of learner model definitions, the overlay learner model and the differential learner model [Kay, 2000]. Kay defines differential modelling as representing a subset of the domain knowledge, and it is only this subset that a student model will deal with. Differential learner modelling is defined in CAVIAr using the learner stereotype goal construct. The course creator must ensure that all the learning goals, defined for a given courseware, can be achieved by all learners. In the OCL constraint in listing 6.13, we have outlined an invariant that ensures that all goal concepts for all learners are covered somewhere in the courseware.

Listing 6.13: OCL invariant ensuring that the union of all learner stereotype goal concepts are covered in the courseware

```
Context CAVIAr
    inv all_learner_goals_covered: self.learningContext.ls->
        iterate(x:LearnerStereotype;
            a:Set(Concept)= Set{} | x.constraints->
            select(oclIsOfType(Goal))
            ->iterate(i:Goal; res:Set(Concept)=Set{}|i.competency.c)->union(a))
        - self.getCoursewareTopics()
            ->iterate(j; res2:Set(Concept)=Set{}|res2->union(j.getTopicConcepts()))
        = Set{}
```

The OCL rule, in listing 6.13, firstly constructs a set containing all the goal concepts for all the learner stereotypes. The invariant in listing 6.13 does this the same way as the invariant in listing 6.12 constructs a set of the common pre-requisite concepts. In this case the invariant iterates through each of the learner stereotypes and adds any goal concepts found to a set, set G. This set is then returned and compared with the set of all concepts covered in the courseware, set C. If there are concepts that are goal concepts and not one of the concepts covered by the courseware the invariant is invalid, i.e. the invariant states $G - C = \emptyset$ must be true.

6.6.3 Validating Courseware Adaptivity

As outlined in section 5.6, CAVIAr provides for courseware adaptivity by allowing the course creator to specify an entry constraint on Courseware Model *topics* in the courseware, where the entry constraint is specified as a learner constraint, learners that satisfy the entry constraint may enter the courseware *topic*.

Courseware validation can be used to ensure that the courseware adapts to a variety of different types of learners in a certain way. For example, validation can check that each topic has supplementary material for learners who are struggling with a concept covered in the courseware. In order to define this OCL constraint we must define what a "struggling learner" is, and what "supplementary support" means. For the purposes of this work we define a "struggling learner" as a learner who has taken a courseware topic that covers some concept and is deemed to have a knowledge level of less than or equal to 0.3 in that concept. We define "supplementary support" for this learner as the provision of additional LOs that cover the said concept and that have a low or very low semantic density, which is delivered after the main topic that is designed to teach the said concept to the learner.

Listing 6.14: OCL invariant insuring the existence of support material for learners struggling with a concept covered in a topic

```
Context Topic
    inv struggling_learner_supported:
        let sameConceptTopic : Topic = self.sequencedAfterTopics()
            ->select(getTopicConcept()=self.getTopicConcept())->first()
        in
            self.sameConceptTopic.entryConstraint.lessThanCompetency.level <= 0.3
            and
            sameConceptTopic.resources
                ->select(oclIsOfType(LO))->forAll(educational.SemanticDensity < Scale::MEDIUM)
```

To ensure that this type of adaptivity is provided the course creator needs to define a constraint to check for the existence of two topics, both covering the same concept and are also sequenced after one another, e.g. t_2 sequenced after t_1. t_1 has no entry requirements, while t_2 is only made available to learners who are struggling on the topic concept(s), where the topic concept(s) is the concept(s) covered by all LOs associated with a given topic, defined in the *getTopicConcepts()* operation in table 6.2. This will ensure that there is supplementary material made available for each concept, covered in the courseware where a learner is struggling with that concept. We have defined this as an OCL constraint in listing 6.14.

6.7 Chapter Conclusion

This chapter has illustrated a constraint-based approach to courseware validation. Constraints are defined in OCL by constraining the allowable CAVIAr Courseware Model and Learning Resource Model structures. This type of validation is focused on the compositional structure of the Courseware Model, rather than its runtime behaviour.

We outlined a classification system for the different types of validation that CAVIAr lends itself to. These were:

- Validation Prerequisites - Used to ensure the data needed for validation is available in the CAVIAr models.
- Validation on Courseware Model Only - Validates Courseware Model elements and data derived from the Courseware Model. This type of validation does not use any of the data in the Learning Context Model.
- Validation Using the Learning Context Model - Validates courseware using the courseware construction concerns defined in the Learning Context Model.

The chapter also looked at the pragmatics of using a constraint language that is designed for software engineers in courseware validation. We accept that only very specialised course creators will be able to define a validation model using OCL. Therefore in an effort to make Validation Model definition more accessible to the majority of course creators we have attempted to raise the level of abstraction at which the course creator defines the Validation Model at. To do this we applied two approaches, OCL helper operations and model-driven OCL generation, in sections 6.2.1 and 6.2.2 respectively. OCL helper operations provide the course creator with OCL functions to encapsulate the most common and/or complex OCL used in defining a CAVIAr Validation Model. The course creator defines the Validation Model using these CAVIAr-specific helper operations. Model-driven OCL generation is a MDE approach to defining a CAVIAr Validation Model. The course creator defines the CAVIAr Validation Model using a Validation Model DSML. This DSML is then used to generate the Validation Model OCL.

In the next chapter we will outline a validation process to build up the CAVIAr models and validate courseware. In chapter 8, we will show how OCL validation, as outlined in this chapter, can integrated into a courseware construction software tool. Our validation approach is evaluated in chapter 9.

Chapter 7

Courseware Validation Process and Activities

7.1 Introduction

In this chapter we present the courseware validation process and its activities. The chapter demonstrates how one goes about validating courseware in a model-based environment. The main aim of the courseware validation process is to provide a structured approach to the course creator to validate constructed courseware using CAVIAr. To enhance the reader's understanding of what is involved in each validation activity, we will exemplify each activity in the validation process using an example case-study course - "Introduction to Databases". The "Introduction to Databases" course is based on a course delivered in Dublin City University (DCU). The learning goals of this course are to teach the basics of database design to computing undergraduates. The experience reported from the example case-study would be mirrored in any other well-defined domains. We anticipate our validation approach to be generally applicable to technical subjects, such as science and engineering, and also structured subjects such as language learning.

The chapter begins by defining the requirements for the case-study course, in section 7.2. Section 7.3 then presents an overview of the CAVIAr validation process. Following this overview, we will examine each of the activities in the courseware validation process in detail with section 7.4 describing how a Domain Model is determined, section 7.5 illustrates how to define a Learning Context Model, section 7.6 shows how to determine a Validation Model for courseware validation and section 7.7 details how a packaged courseware definition is converted to a CAVIAr Courseware Model for validation. We outline what happens at validation time in section 7.8 and after validation is complete in section 7.9.

In each section that covers a courseware validation activity we outline the principles of that activity, specifying what the activity aims to achieve and how. We also exemplify the validation activity principles using the case-study course. We conclude this chapter with a summary of the validation process in section 7.10.

7.2 Example Case-Study Course Specification

In our example case-study, the course creator must create a course explaining the elementary concepts of database theory and design. The course descriptor we used for this is DCU module CA218 - "Introduction to Databases"[1].

The course accreditation body - Dublin City University (DCU), has defined a module descriptor. The module descriptor defines the learning goals of the course and any other pedagogical constraints, such as the presumed learner knowledge before starting the course as well as learner assessment requirements.

According to the course descriptor the module aims are described as:

> "To provide students with an introduction to and an overview of database systems including database design, Entity Relationship data modelling, the relational model of data and SQL, as well as an overview of some database products."

The learning outcomes as defined in course descriptor are:

> "As a result of this course, students will be in a position to design and implement a database for real world application, covering all stages of the [database

[1] `http://www.dcu.ie/registry/module_contents.php?function=2&subcode=CA218`

design] process from conceptual design and layout through to writing SQL for transactions. Students will also be very familiar with SQL."

The learning outcomes indicate that not only will the learner require conceptual knowledge, but will also require skills knowledge, to "design and implement a database". The course creator must therefore ensure that the courseware provides skills knowledge for the required skills.

The module descriptor defines the course pre-requisites as "none". DCU defines a pre-requisite as "any other modules offered by the University in which a student must have achieved satisfactory performance before enrolling on a particular module"[2].

Courseware developed for CA218 - Introduction to Databases must adhere to the following constraints:

- The total timing of all learning material is outlined as follows:
  - Lecture material - 24 hours
  - Tutorial material - 12 hours
  - Laboratory material - 12 hours

In the example case-study the course creator has elected to specify the following additional constraints to be applied to the courseware to improve the instructional design of the courseware:

- All courseware topics must contain at least one lecture LO.
- LOs used in each topic must just address the concerns of the topic (i.e. the LO must be annotated with the concept the topic addresses).

An indicative syllabus has also been defined by a domain expert based on the module descriptor. The indicative syllabus is defined as an unordered collection of course concepts. The domain expert has specified the importance of some of the concepts listed as they are mentioned in the module aims. These concepts are denoted using a bold font, as follows:

[2]`http://www.dcu.ie/registry/examinations/standards.shtml`

- Information Systems
- **Database Overview**
- Storage Structures
- **Entity-Relationship Data Modelling**
- **Relational Model of Data**
- **SQL**
- **Overview of Database Products**
- The System Catalogue
- Views
- Database Design & Normalisation
- Web Databases and JDBC

The course creator has also defined the following conceptual sequencing constraints:

- Information Systems must be sequenced before all other courseware topics.
- Database Overview must be sequenced before all other courseware topics except Information Systems.
- SQL must be sequenced before Views, as Views are a type of SQL.
- Database Design & Normalisation must be sequenced after the Relational Model of Data.

The course creator also wishes to ensure that the courseware developed correctly implements Reigeluth's Elaboration Theory [Reigeluth, 1999b].

There are two types of learner who take the databases course, Computer Applications Information Systems (CAIS) students and Computer Applications Software Engineering (CASE) students. The CAIS students do not need to learn about "Storage Structures".

CAIS students are also guaranteed to have done a module in Information Systems in year one of their qualification course.

Based on this course specification a courseware has been developed by the course creator. To verify that the courseware adheres to the course requirements, described in this section, pre-delivery, the course creator must validate the courseware. To validate the courseware using CAVIAr the course creator must build up the CAVIAr models and import the courseware package for the CA218 courseware for validation and then perform validation. In the following sections we will outline a validation process that can be used to validate courseware.

7.3 Validation Process Overview

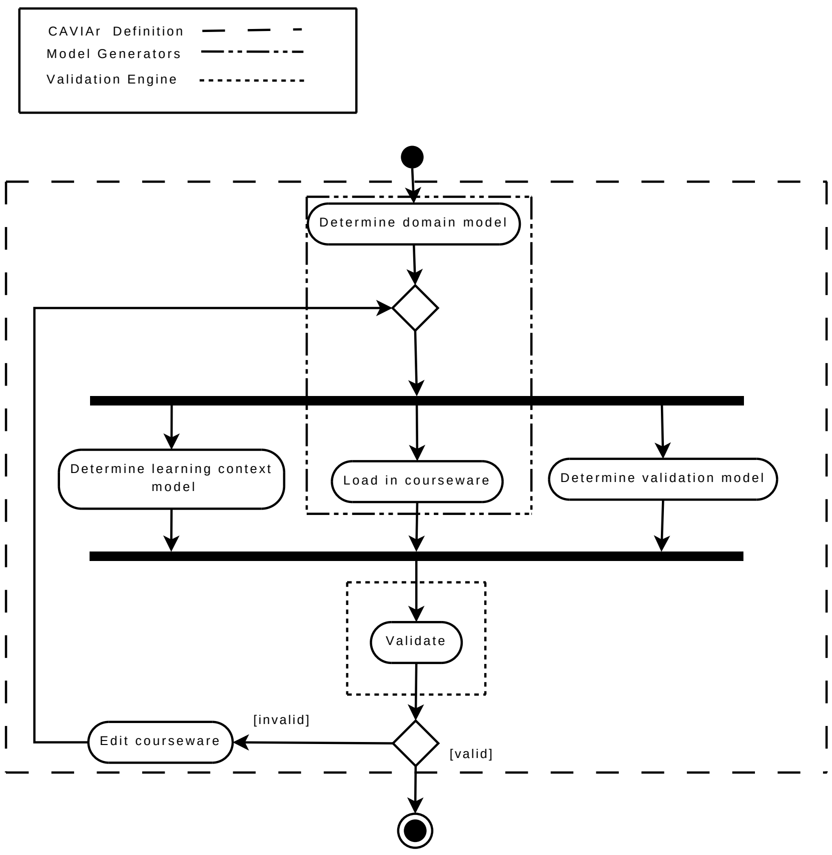


Figure 7.1: UML Activity Diagram of the Methodological Framework

Our approach to validating courseware consists of five activities. These activities are summarised in the UML activity diagram in figure 7.1. We have used UML activity diagrams in this chapter to communicate process flow as it is a common notation for defining a process flow in software engineering and has a well defined metamodel [OMG, 2007]. The process starts with the assumption that the constructed courseware is described using either IMS LD or ADL SCORM, and LOs in the courseware are annotated using IEEE LOM metadata. In the following sub-sections we present in detail each of the five activities. The five activities in the validation process are as follows.

- Determine the subject Domain Model for courseware.
- Determine a Learning Context Model in the context of the Domain Model.
- Determine the courseware Validation Model.
- Render the courseware definition as a CAVIAr Courseware Model.
- Validate the courseware.

The validation process is supported by the following:

- CAVIAr definition - see chapter 5.
- CAVIAr model generators - see chapter 8.
- Courseware validation engine - see chapter 6.

7.4 Determining the Domain Model

The initial step in the courseware validation methodology is to locate a definition for the subject domain that the constructed courseware teaches. The course creator must determine a subject domain model to use. The course creator can either define a subject domain model from scratch or reuse an existing domain model definition, perhaps in the form of a Semantic Web ontology, as in [Melia et al., 2005, Holohan et al., 2005, Jovanović et al., 2005, Aroya et al., 2002, Yang et al., 2005]. In our work we assume that the course creator reuses the subject domain model that is also used to classify LOs used by the courseware. Should the course creator use a different domain model or define their own domain model for validation, this would have to be mapped or merged to the "classification source" used in the LOM classification for the LOs used in courseware [IEEE LTSC, 2002]. The "classification source" in LOM and the domain model definition are ontologies, as defined on the ontological spectrum in [Daconta et al., 2003]. The mapping and merging of ontologies is beyond the scope of this thesis, more details can be found in [Gomez-Perez et al., 2004].

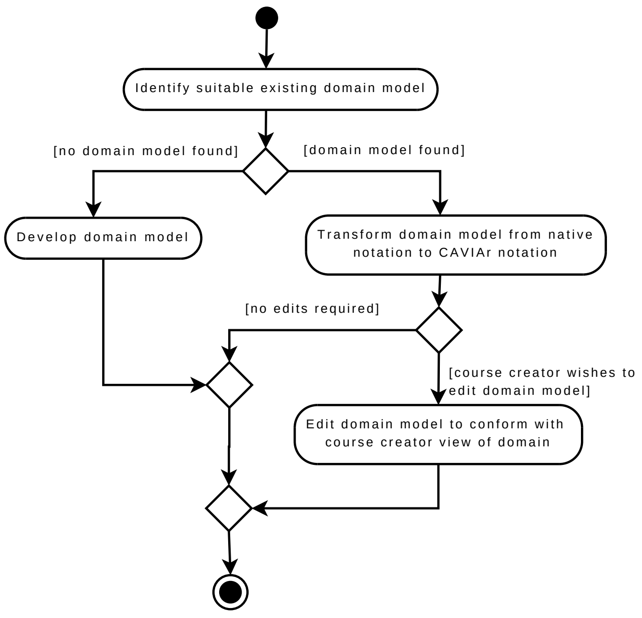

Identify suitable existing domain model
[no domain model found]
[domain model found]
Develop domain model
Transform domain model from native notation to CAVIAr notation
[no edits required]
[course creator wishes to edit domain model]
Edit domain model to conform with course creator view of domain

Figure 7.2: UML Activity Diagram for determining the CAVIAr Domain Model

7.4.1 Developing a Domain Model

If there is no domain model that suits the needs of the course creator, he or she may choose to create their own Domain Model. Tool support for defining a Domain Model is outlined in section 8.6.1. A common approach to defining a new domain model is to use an upper level ontology, such as the Suggested Upper Merged Ontology (SUMO) [Niles and Pease, 2001] or OpenCyc ontology [Lenat, 1996].

7.4.2 Incorporating an External Domain Model

In most instances where courseware validation takes place, the course creator will use an existing domain model. A domain model is firstly identified and then transformed into a CAVIAr Domain Model. Any external domain model definition can be used in CAVIAr as long as the following conditions are met:

- The domain model has a formal metamodel or abstract syntax.
- The domain model's metamodel can be mapped and transformed to the CAVIAr Domain Model metamodel.

For the domain model to be used in CAVIAr it is transformed from its native representation into a CAVIAr Domain Model. The following assumptions are made about the domain models used in CAVIAr:

- The domain model has a taxonomy of concepts, the scope of concepts becomes more specialised or generalised as you move up and down the taxonomy.
- The domain model is self-contained, in that the domain model does not reference any external resources.
- There is no pedagogical information in the domain model.
- There are no circular dependencies or relationships within the domain model.
- There are no orphan concepts in the domain model.

Once the external domain model definition has been transformed to a CAVIAr Domain Model the course creator has the option of editing the Domain Model to adapt it into his or her view of how the domain is conceptually structured.

7.4.3 Case-study

In our case-study, the course creator takes a domain model definition, defined in SKOS, and transforms it into a CAVIAr Domain Model representation. These are simple syntax transformations where a syntactical element from one notation is mapped to a syntactical element of the other notation.

In our example case-study, the course creator chooses to use a SKOS ontology that describes the databases domain. SKOS is a common Semantic Web technology used for defining simple knowledge structures. We envision the Semantic Web to be one of the primary sources of knowledge structures as common shared conceptualisation, such as those found in ontologies, form the backbone of Semantic Web technologies [Stollberg et al., 2006].

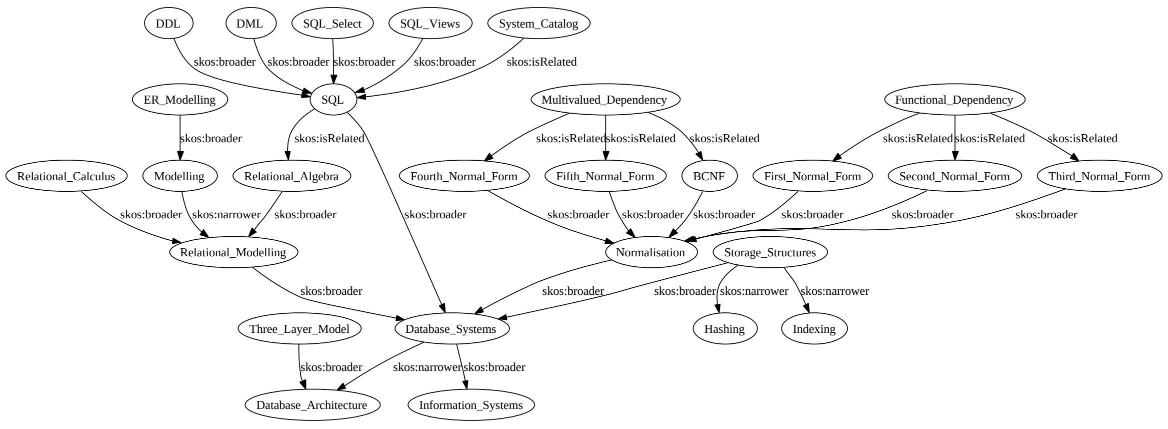


Figure 7.3: Databases domain model defined using SKOS

In figure 7.3 we outline the domain model defined for the purposes of validating the CA218 course. This domain model provides a sufficient domain knowledge structure required for CAVIAr courseware validation (see appendix A for SKOS XML binding).

This domain model has a central concept, *Database_System*. The *Database_System* concept is broken into six more specialised concepts in the domain model using the SKOS *skos:narrower* and *skos:broader* relationship, which are *information_systems*, *Normalisation*, *Database_Architecture*, *Storage_Structures*, *SQL* and *Relational_Modelling*. These concepts are then broken down even further using the SKOS *skos:narrower* relationship and its inverse relationship, the SKOS *skos:broader* relationship creating a taxonomy of database concepts. The other relationship used in the domain model is the SKOS relationship *skos:related*. This allows the representation of concepts that are related to each other but are not broader/narrower than each other. For example, the concept *Third_Normal_Form* and *Functional_Dependency* are related, as the third normal form is based on the concept of functional dependency.

The SKOS syntax is mapped to the CAVIAr abstract syntax as defined in section 5.3.3, creating a CAVIAr Domain Model for database systems.

7.5 Defining the Learning Context Model

After the Domain Model is determined, the Learning Context Model can be defined in the context of the Domain Model. The Learning Context Model is defined by specifying additional modelling constructs in the context of the Domain Model. It consists of learner stereotype definitions and pre-requisite relationships between concepts in the Domain Model. Each learner stereotype definition consists of the courseware goals and presumed knowledge of that learner stereotype. Both are defined as knowledge elements in terms of the Domain Model concepts.

In figure 7.4 outlines the tasks involved in defining a CAVIAr Learning Context Model using a UML Activity Diagram.

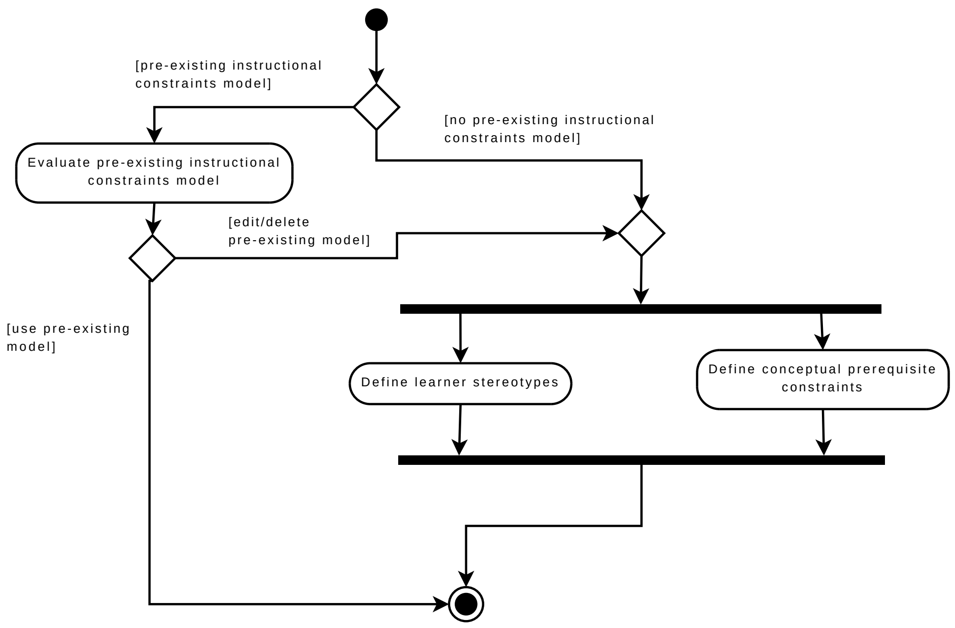

[pre-existing instructional constraints model]
[no pre-existing instructional constraints model]
Evaluate pre-existing instructional constraints model
[edit/delete pre-existing model]
[use pre-existing model]
Define learner stereotypes
Define conceptual prerequisite constraints

Figure 7.4: UML Activity Diagram outlining the activities for determining the CAVIAr Instructional Constraints Model

7.5.1 Defining the Conceptual Pre-requisite Constraints

After the Domain Model concepts have been defined, the course creator may impose sequencing constraints on the ordering of concepts covered in the courseware. This is done by specifying pre-requisite constraints. The course creator specifies a pre-requisite relationship where one Domain Model concept is the source of the relationship and a different Domain Model concept is the relationship's target. This means that the Courseware Model must be sequenced in such a way where the pre-requisite relationship's target concept is sequenced before the relationship's source concept.

7.5.2 Defining Learner Stereotypes

In CAVIAr the course creator must define the anticipated types of learners that he or she envisions will use the constructed courseware. Each learner stereotype is defined in terms of its goal concepts and its presumed knowledge entering the courseware. In order to define a learning context, the course creator must decide on what learner stereotypes he or she wishes to define. The simplest and quickest approach to defining learner stereotypes is to define one learner stereotype that all learners fit into. In this case the course will not discriminate at all between learners, a "one size fits all" approach to courseware design. Although this is the simplest to develop it does not consider the different pedagogical needs of learners who take a course [Brusilovsky et al., 1998]. At the other extreme, a learner stereotype can be designed for each learner allowing for very personalised courseware validation, but is very labour-intensive to develop. Using learner stereotypes the course creator may distinguish between different groups of learners. For example learner stereotypes could be used to distinguish ability levels in a group of learners, or possibly to distinguish academic and/or professional backgrounds.

The course creator then defines learner stereotypes in CAVIAr accordingly. For each of the learner stereotypes defined in CAVIAr the course creator must define the course goal knowledge and presumed learner knowledge.

7.5.2.1 Defining the Course Goals

Course goals are defined in terms of knowledge elements, by specifying a Domain Model concept, a knowledge level and a knowledge type required for a learning goal, for a specific learner stereotype. When a concept is defined as a goal concept for courseware, that Domain Model concept is essentially tagged as a goal for a particular learner stereotype, with an expected knowledge level and type of knowledge, as outlined in section 5.4.

The goal concepts are derived from the module specification, such as that outlined in section 7.2. This is done by identifying the concepts that capture the learning goals for each learner stereotype and defining the knowledge type and level required for that concept. This is not a trivial task and is open to much interpretation by the course creator. The course creator must take the information supplied in the course descriptor and formulate from it the CAVIAr knowledge elements. This is done by following the steps below:

1. Identify the key knowledge concepts in the course descriptor.

2. Identify the type of knowledge required, is the knowledge required conceptual knowledge or skills knowledge or both.

3. Identify the level of the knowledge required for each knowledge concept and level.

4. Map these elements to CAVIAr knowledge elements (knowledge elements are defined in section 5.4).

CAVIAr also allows the course creator to specify alternative goal concepts, and to import course goals from other Learning Context Models and integrate them into a given learning context. If the course creator wishes to reuse a learning goal the course creator simply *includes* the pre-defined learning goal in the learning goal he or she is defining, using the goal *PartOfRelationship* defined in section 5.4. Should the course creator wish to define one learning goal as an alternative to another learning goal, the goal are related via the *AltGoal* goal relationship, also defined in section 5.4.

7.5.2.2 Defining the Presumed Learner Stereotype Knowledge

The presumed learner stereotype knowledge allows the course creator to define the knowledge he or she expects a particular learner stereotype to have upon starting the courseware. Again, this is done in the context of the Domain Model by specifying the concepts the learner should know. The course creator must again define the knowledge type and level for the presumed knowledge concepts. This involves assessing the knowledge a typical learner in a particular stereotype will have in terms of the knowledge type and knowledge level for a specific Domain Model concept.

Presumed knowledge is added to the CAVIAr model through a relationship between the learner stereotype's *PresumedKnowledge* class and a *KnowledgeElement* that references the Domain Model concept. This in effect tags the concept as presumed knowledge for a specific learner stereotype.

7.5.3 Case-Study

Our case-study continues from section 7.4.3. Here we will exemplify the activities involved in defining a Learning Context Model by outlining how the course creator defines a Learning Context Model for the databases courseware.

7.5.3.1 Expressing the Case-study in terms of the CAVIAr Instructional Constraints Model

Some instructional design constraints in the case-study course's module descriptor, in section 7.2, can be expressed in the CAVIAr Learning Context Model. Here, we firstly list the constraints from the module descriptor defined in section 7.2, as an informal list. We then define the constraints as CAVIAr learning context knowledge elements in table 7.1.

- The learner stereotypes defined for the courseware are:
  - CASE - Computer Applications - Software Engineering students
  - CAIS - Computer Applications - Information Systems students

- We can also list a number of goal concepts for CAIS and CASE learners. The goals of the course are applied to both learner stereotypes unless otherwise stated and can be defined as follows:
  - Each element in the indicative syllabus is defined as a required mid-level conceptual goal.
    - Only CASE students require knowledge on “storage structures”.
  - For concepts elicited from the module aims (bold in the indicative syllabus in section 7.2) learners must achieve a high level of knowledge.
  - The learning outcome specifies the learner must be able to “design and implement a database”. The course creator is able to identify skill concepts needed to be able to do this and specifies these concepts as mid-level skills knowledge goals. Theses are:
    - ER Data Modelling
    - Normalisation
    - SQL - for database definition (DDL)
  - The course creator recognises that SQL is not only important in the design and definition of databases systems but also in manipulating a working database, this too is defined as skills knowledge as the learning outcomes state that learners must be “very familiar with SQL”, and be able to write SQL for transactions. Therefore a high level of skills knowledge is required on the SQL concept.
- The course creator must also define the knowledge we expect the learner to enter the courseware with. To do this the course creator simply assesses what is covered in the learner’s previous modules. With this in mind we have defined the following anticipated knowledge competencies for the typical CA218 learner:
  - We expect all learners to have mid-range conceptual knowledge on basic information systems, therefore the *Information_Systems* concept is a concept which

all learners are expected to have a mid degree of conceptual knowledge before taking the course. CAIS students have additional lectures in *Information_Systems* in year 1, and therefore have a slightly higher conceptual knowledge level in this concept.

- The following conceptual pre-requisite constraints are defined on the databases course specification as outlined in section 7.2. It should be noted that these are conceptual pre-requisite constraints and are defined for all learner stereotypes:
  - Mid-level conceptual knowledge in *Information Systems* is a pre-requisite for all other goal concepts.
  - Mid-level conceptual knowledge in *Database Overview* is a pre-requisite for all courseware concepts except the *Information Systems* concept.
  - Mid-level skills knowledge in *SQL* is a pre-requisite for the *views* concept, as views are a type of SQL.
  - Mid-level conceptual knowledge in the *Relational Model of Data* are pre-requisites to the *Database Design & Normalisation* concepts.

From the requirements listed above the course creator can define the Learning Context Model for the databases course. Table 7.1 defines all the facets of the example case study learning context model.

7.6 Determining the Validation Model

The OCL-based CAVIAr Validation Model definition is described in chapter 6. The course creator can use the various types of constraints defined in this chapter to specify courseware validation rules. The constraint rules outlined in chapter 6 can be aggregated to ensure the correct application of an instructional design theory. This is to say that ensuring the correct application of an instructional design theory can be defined as part of a CAVIAr Validation Model using a set of instructional constraints that courseware must adhere to. In this section

Table 7.1: Definition of anticipated learner knowledge defined in terms of CAVIAr knowledge elements

Concept	Knowledge level	Knowledge Type
CASE Student - Goals		
SQL	0.7	skill
Normalisation	0.7	skill
Normalisation	0.7	conceptual
Data Modelling	0.5	skill
Information_System	0.5	conceptual
JDBC	0.5	skill
Three_Layer_Model	0.5	conceptual
Database_Architecture	0.5	conceptual
Storage_Structures	0.5	conceptual
CASE Student - Presumed Knowledge		
Information_Systems	0.3	conceptual
CAIS Student - Goals		
SQL	0.7	skill
Normalisation	0.7	skill
Normalisation	0.7	conceptual
Data Modelling	0.5	skill
Information_System	0.3	conceptual
JDBC	0.5	skill
Three_Layer_Model	0.5	conceptual
Database_Architecture	0.5	conceptual
CAIS Student - Presumed Knowledge		
Information_Systems	0.7	conceptual

we outline how a CAVIAr Validation Model is defined to ensure the correct application of an instructional design theory in courseware.

Figure 7.5 outlines the steps involved in defining an instructional design theory in the form of CAVIAr Validation Model constraints. These constraints can be used to ensure the correct application of an instructional design theory in the constructed courseware. This process begins with the course creator deciding on the instructional design theory to use and then investigating the possibility of reusing a Validation Model that validates the application of the chosen instructional design theory in courseware. If one is found, it can be reused (and possibly extended). This minimises the Validation Model definition effort for the course creator. However if a Validation Model cannot be reused the course creator must define a Validation Model. To achieve this the instructional design theory is broken down into instructional constraints that must be satisfied for the instructional design theory to be implemented correctly. To define the instructional constraints, the course creator must identify the main principles of the instructional design theory being used. Each principle identified must then be converted into instructional constraints that must be true for the principle to be applied correctly in the courseware. The course creator then defines each constraint in terms of the CAVIAr metamodel. CAVIAr constraints are formalised using the CAVIAr constraint language, OCL.

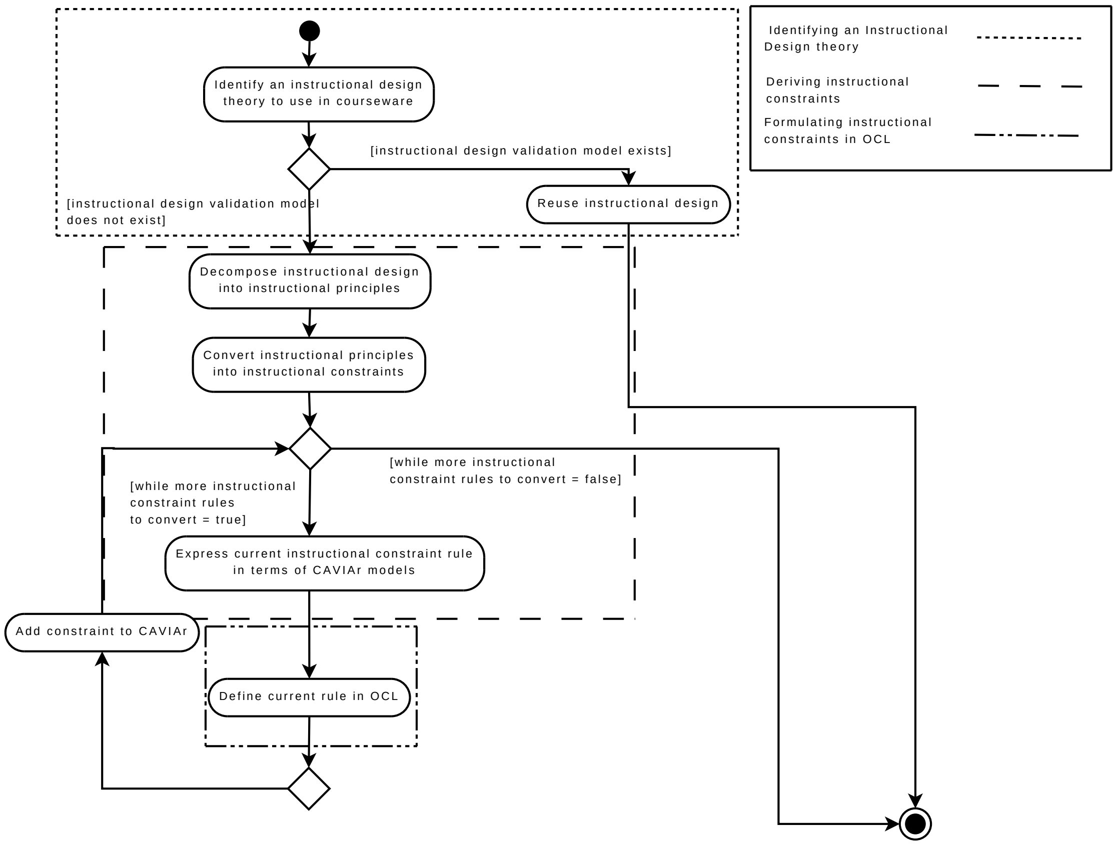

Identifying an Instructional Design theory
Deriving instructional constraints
Formulating instructional constraints in OCL
Identify an instructional design theory to use in courseware
[instructional design validation model exists]
Reuse instructional design
[instructional design validation model does not exist]
Decompose instructional design into instructional principles
Convert instructional principles into instructional constraints
[while more instructional constraint rules to convert = false]
[while more instructional constraint rules to convert = true]
Express current instructional constraint rule in terms of CAVIAr models
Add constraint to CAVIAr
Define current rule in OCL

Figure 7.5: UML Activity Diagram detailing the definition of the CAVIAr Validation Model

In this section, we will look in detail at the activities involved in defining the CAVIAr Validation Model. We have divided the Validation Model definition process into three main parts, which we will examine in the following subsections:

- Determining the instructional design used in courseware (principally determined by the instructional designer).

- Deriving informal instructional constraints from the instructional design of choice (normally derived by the instructional designer and instructional designer programmer) and forming CAVIAr-based constraints based on the informal instructional constraints.
- Defining instructional constraints in OCL (principally defined by the instructional designer programmer).

We will then look to our running case-study to exemplify the Validation Model definition process.

7.6.1 Determining the Instructional Design

Instructional design theory defines how knowledge should be taught to a learner given a learning scenario. The course creator defines the instructional design to ensure instruction is as effective and efficient as possible in facilitating the learner from an initial knowledge state to the learning goal knowledge state. A course creator may apply their own instructional design or may apply one of the many instructional design theories in the literature [Reigeluth, 1999a, Gagné et al., 2005].

To validate the application of an instructional design theory the course creator checks if there is a CAVIAr Validation Model that can be reused, which validates the application of the instructional design theory that the course creator has attempted to apply in courseware. If such a Validation Model exists the course creator can reuse this Validation Model and if not the course creator must define his or her own Validation Model.

7.6.2 Deriving Instructional Constraints

In validating courseware, the course creator looks to confirm that an instructional design theory has been applied correctly in courseware. To validate the courseware against an instructional design theory, the course creator must break down the theory into instructional constraints.

An instructional design theory can be defined as a set of instructional principles, as demonstrated by Reigeluth in his outline of each instructional design theory covered in [Reigeluth, 1999a]. For an instructional design theory to be used correctly, the course creator must adhere to these principles. In defining the requirements of adhering to instructional principles the course creator can define instructional constraints based on these principles that must be true for the correct application of the instructional design theory. Once the instructional constraints that make up the definition of an instructional design theory have been defined, the course creator specifies each constraint in terms of the CAVIAr metamodel. The output of this step is a series of constraints defined informally in terms of the CAVIAr metamodel.

7.6.3 Formulating Instructional Constraints in OCL

At this stage the instructional design has been broken up into instructional constraints that are defined in terms of CAVIAr. These constraints must then be used to formulate a CAVIAr Validation Model using OCL. In chapter 6, we have described how constraints are defined. The instructional designer programmer takes each of the constraints and defines them in terms of the CAVIAr metamodels and expresses them formally in OCL.

7.6.4 Case-study

The Validation Model is used to define rules on the Courseware Model that must be adhered to for courseware to be deemed valid. The Validation Model provides the course creator with the facility to express rules and constraints using CAVIAr metamodel definitions.

The first step in defining the Validation Model for our case-study courseware is to establish the instructional design definition that the course creator wishes to validate courseware against. In the case-study this is defined by establishing the instructional design theory the course creator has attempted to implement and any other additional requirements defined in the course criteria (section 7.2). From the course criteria we establish the following courseware instructional design requirements:

- The courseware must be designed as specified in Reigeluth's Elaboration Theory.
- All courseware topics must contain a "lecture" LO.
- The LOs used in topics must focus on what that topic is covering.
- All LOs of type "lecture" must not exceed 1440 minutes (24 hours).
- All LOs of type "experiment", "simulation", "exercise" and "self_assessment" must not exceed 1440 minutes (24 hours) as these are collectively the tutorial and laboratory material.

To define constraints for the elaboration theory, the course creator checks to see if a Validation Model already exists that checks for the correct application of the elaboration theory in courseware. If one exists it can be reused. In our case-study no such Validation Model exists. The course creator must develop a Validation Model that ensures the application of the elaboration theory in the Courseware Model.

The elaboration theory is firstly broken down into its key instructional principles, which must be true for the theory to be applied correctly. These principles have been defined by Reigeluth in [Reigeluth, 1999b] (principles are numbered P1-5):

- P1: Tasks, concepts and principles are arranged from simple to complex, starting with the simplest real-world version of the task, concept or principle moving to ever more complex versions.
- P2: Pre-requisite relationships between knowledge elements taught in the course [Reigeluth, 1999b, p431] must be respected in course sequencing.
- P3: Teach supporting content together with the concept, principles or tasks that it is most related to.
- P4: Group tasks, concepts and principles together with supporting content into learning episodes.
- P5: Give learners some choice as to what to "elaborate" first/next.

Once the instructional design has been formulated into instructional principles we can define each as an instructional constraint that must be true for the elaboration theory to be applied correctly. We do this as follows. Constraints are numbered C1-5. The instructional design principle on which a given constraint is based is stated at the end of the constraint:

- C1: Courses must start with the simplest topics and progressively allow for more complex or specialised topics to be delivered (P1).

- C2: When sequencing course topics pre-requisite relationships that exist between knowledge elements must be respected (P2).

- C3: If a LO in a topic specifies a dependency on another LO(s), the other LO(s) is supporting content and must be available in that topic (P3).

- C4: All LOs in a given course topic (learning episode) must be focused on teaching a common knowledge element (P4). This constraint also addresses the course criteria that states "LOs used in each topic must address the concerns of the topic".

- C5: The learner must be able to choose which high-level courseware topic to learn about and when (P5).

We also add constraints for the non-elaboration theory constraints defined in the course criteria section, section 7.2:

- C6: There must be a "lecture" LO for each part of the course.

- C7: The total duration of "lecture" LOs must not exceed 1440.

- C8: The total duration of "tutorial" and "lab" LOs must not exceed 1440.

Each constraint is then defined in terms of the CAVIAr metamodels:

- C1: *Topics* must start with the broadest topic-concept according to the CAVIAr Domain Model, and progressively allow the learner to learn more complex or specialised topic-concepts. The topic-concept is defined by the *getTopicConcept()* OCL operation in section 6.2.1.

- C2: If a topic-concept is the source of a *PRE_REQUISITE ConceptRelationship* there must be a *SEQUENCED_AFTER TopicRelationship* that reflects the conceptual sequencing constraint.

- C3: Should a LO in a task topic be based on another LO(s), "based on" is defined when a LO has a *Relation* where *RelationKind* is *BASED_ON*, both LOs must be in the same *Topic*. The *BASED_ON* relationship between LOs indicates that the target LO in the relationship is "supporting content needed" as defined in the elaboration theory.

- C4: All *LOs* in a courseware *Topic* must be classified with only the topic-concept(s), where topic-concept is defined by the *getTopicConcept()* OCL operation.

- C5: In the *Courseware* construct, there must be less than half the number of contained *SEQUENCED_AFTER TopicRelationships* to *Topics*.

We also add the following constraints for non-elaboration theory constraints defined in section 7.2, as follows:

- C6: There must be a LO with a *learningResourceType* of *LECTURE* in each CAVIAr courseware topic.

- C7: The sum of the duration of all *LECTURE* LOs in a CAVIAr Learning Resource Model must not exceed 1440.

- C8: The sum of the duration of all *EXPERIMENT*, *SIMULATION*, *EXERCISE* and *SELF_ASSESSMENT* LOs ("tutorial" and "lab" LOs) must not exceed 1440 (minutes).

The course creator has now specified the courseware constraints in terms of the CAVIAr metamodels. These constraints are then converted to OCL. In listing 7.1 and 7.2, we have taken the constraints for the databases courseware (C1-8) and defined the corresponding OCL. The mapping from the constraint definition above to each OCL constraint is made explicit through OCL comments.

Listing 7.1: Case study Validation Model - part 1

```
Context Topic

--C1 - Principles , tasks and concepts must be sequenced from simple to complex
    inv courseware_sequenced_from_simple_to_complex:
        self.sequencedAfterTopics()
            ->iterate(y:Topic; a:Set(Concept)=Set{}| a->union(y.getTopicConcepts()))
        - getTopicConcepts()->iterate(x:Concept; a:Set(Concept)=Set{}| a
            ->union(x.narrower())->union(x.siblings()))
         = Set{}

--C2 --Conceptual pre-requisites are respected in topic sequencing
    inv conceptual_prerequisites_are_respected:
        self.getAllTopicConcepts()
        ->iterate(x:Concept; a:Set(Concept)=Set{}| a->union(x.prerequisite(0.5, true)))
        - self.sequencedAfterTopics()
            ->iterate(y:Topic; b:Set(Concept)=Set{}| b->union(y.concepts))
         = Set{}

--C3 - Supporting learning resources must be available in the same topic
    inv supporting_content_available_in_topic:
        resources->select(oclIsTypeOf(LO))
        ->iterate(r:Resource; los:Set(LO)=Set{}| los->union(r.oclAsType(LO)))
        ->iterate(y:LO; a:Set(Resource)=Set{}| a
            ->union(y.metadata.relations->select(kind=RelationKind::BASED_ON))))
        - self.resoures = Set{}

--C4   - LOs must be focused on the topic concept
    inv LO_concept_focus:
        self.getAllTopicConcepts()- self.getTopicConcepts()=Set{}

--C6 - All tasks must have a lecture LO
    inv task_has_lecture_LO:
            getAllTopicLOs()->
                exists(educational.learningResourceType = ResourceType::LECTURE)

context Courseware
    def: getLOType(type:String): Set(LO) =
        self.getCoursewareResources()->select(oclIsTypeOf(LO))
        ->iterate(i:Resource; res:Set(LO)=Set{}|res->including(i->oclAsType(LO)))
        ->select(type)

--C5 - There is only limited number of sequencing constraints for high-level topics in courseware
    inv limited_sequencing_constraints_in_courseware_topics:
        topics->iterate(i; res:Integer=0|res = res+i.sequencedAfterTopics()->size())
        < (topics->size()/2)

--C7 - The sum of all lectures in the courseware must not exceed 24 hours
    inv lecture_time_exceeded:
        self.getLOType('Lecture')
        ->iterate(i:LO; durs:Set(Integer)=Set{}|durs->including(i.technical.duration)
        ->sum() < 1440
```

Listing 7.2: Case study Validation Model - part 2

```
context Courseware
--C8 - The sum of the tutorial LOs in the courseware must not exceed 24 hours, tutorial LOs are defined
        as LO of type experiment, exercise, simulation or self assessment
    inv lab_or_tutorial_time_exceeded:
        let los:Set(LO)=
        self.getLOType('experiment')->union(self.getLOType('exercise'))
            ->union(self.getLOType('simulation'))->union(self.getLOType('self_assessment'))
        ->iterate(i:LO; durs:Set(Integer)=Set{}|durs->including(i.technical.duration)
        ->sum() < 1440
```

7.7 Rendering the Courseware Definition into CAVIAr

In order for courseware to be validated in CAVIAr it must be converted into a CAVIAr Courseware Model. In TEL courseware is described using a courseware specification such as SCORM [ADL, 2004] or IMS LD [IMS, 2003b]. In order to integrate CAVIAr validation with the state of the art, it is necessary to be able to translate the courseware specification into a CAVIAr Courseware Model. In figure 7.6, we outline the steps involved in rendering a courseware specification ready for validation. That is converting the XML representation of a courseware definition, where a courseware is defined using a courseware specification and a set of LO metadata descriptions, from its native representation and transforming it into CAVIAr models.

Much of this process is automated and transparent to the course creator. Implementation level details on how the courseware specification is transformed into CAVIAr can be found in section 8.8. For the course creator to render a courseware specification into a CAVIAr Courseware Model, all he or she must do is choose the correct transformation that will transform the XML courseware specifications used into a CAVIAr Courseware Model.

7.7.1 Case-study

In our case-study, the databases courseware has been defined using the ADL SCORM 2004 specification. SCORM is described, in detail, in section 2.2.6. LOs used in the courseware are annotated using the IEEE LOM metadata standard (described in section 2.2.3).

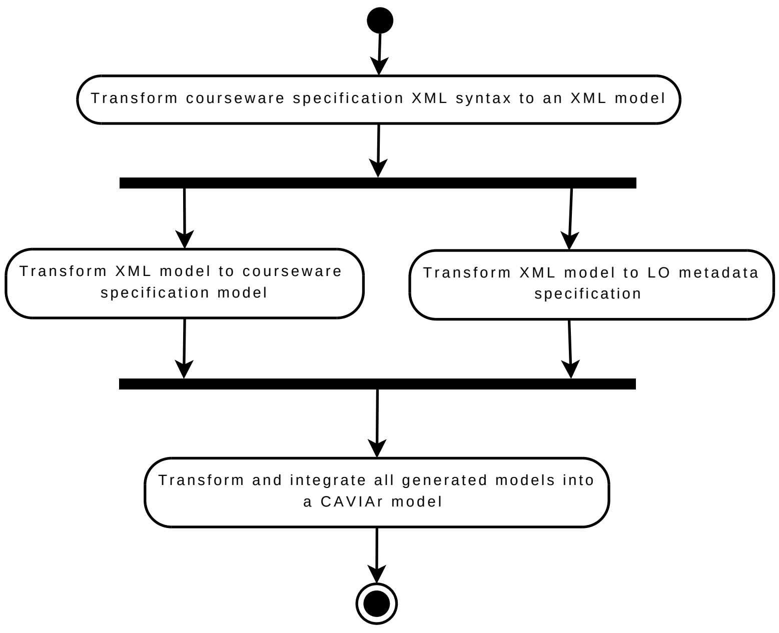

Transform courseware specification XML syntax to an XML model
Transform XML model to courseware specification model
Transform XML model to LO metadata specification
Transform and integrate all generated models into a CAVIAr model

Figure 7.6: UML Activity Diagram outlining the steps involved in loading a Courseware Model into the validation framework

A transformation process is defined to transform the SCORM specification and LOM standard, used for LOs in the courseware, into CAVIAr. This process is a co-ordinated series of transformations and managed using a transformation orchestration script. The process transforms the SCORM specification into a CAVIAr Courseware Model and the LOM definitions into a Learning Resource Model. These CAVIAr models are then integrated together. This process is transparent to the course creator.

7.8 Validation of Courseware

Validation of courseware takes place when the courseware and the courseware's learning context are modelled using CAVIAr and the courseware Validation Model has been specified. In order to validate the courseware a validation engine, in the form of an OCL checker, is used to check each of the Validation Model constraint rules is valid for the constructed courseware.

In this section, we will look at how to use OCL checker tools to allow for the validation of a CAVIAr Courseware Model.

7.8.1 Validation Engine Initialisation

The validation engine must be made CAVIAr-aware, this is done by loading the CAVIAr metamodels into the OCL checker. Constraints are then specified in terms of the CAVIAr metamodels and loaded into the OCL checker. The CAVIAr models are then loaded into the OCL checker, which checks that the OCL constraints defined are satisfied in the Courseware Model if any part of the Courseware Model violates an OCL invariant, the constraint and courseware modelling construct is flagged and the course creator is notified.

7.8.2 Validation Outcome

Typically an OCL checker will run through each of the invariants stated in the Validation Model and test each one against the CAVIAr model. Should an invariant be deemed invalid, the invariant is flagged. The course creator is made aware of the failed invariants once validation is complete and all invariants have been checked against the Courseware Model.

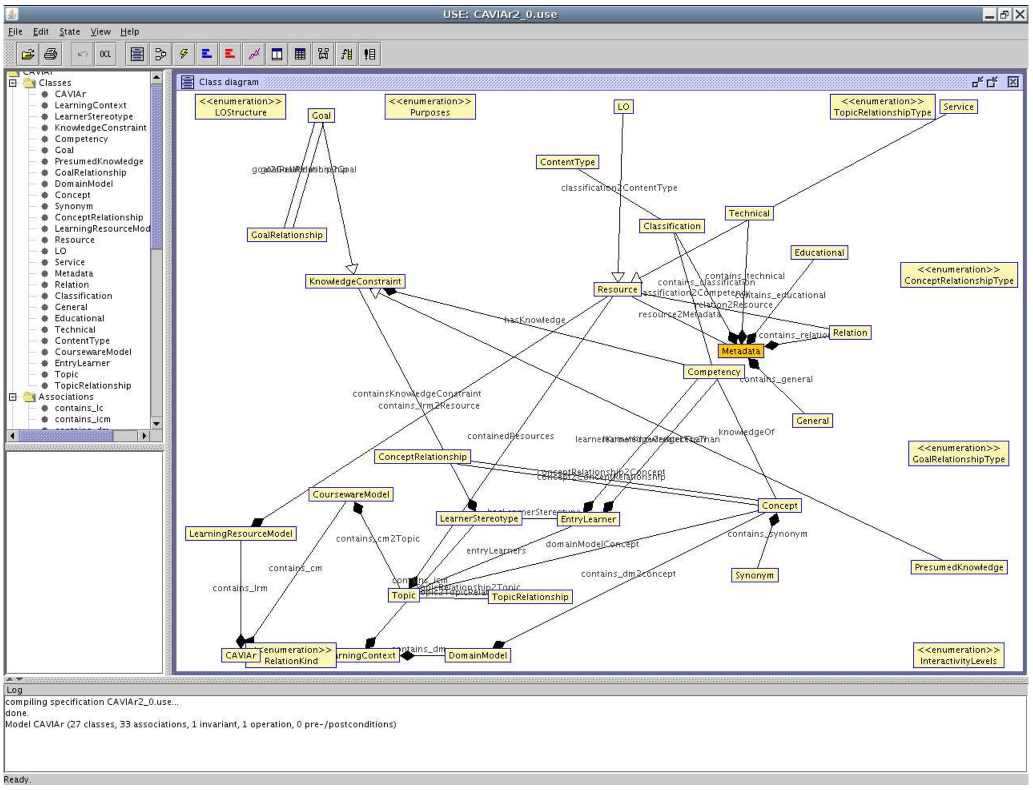
It is not possible to make recommendations to the course creator on corrective measures for invalid courseware using an OCL checker. An OCL checker can only give a boolean value, stating whether or not an invariant passed or failed. We mentioned in section 2.4 that OCL can be used as a model query language. The model element(s) that caused a constraint to fail can be retrieved by redefining the OCL invariant definition into a model query.

The burden of analysing the problems identified through validation rests with the course creator, he or she must assess the problems in the CAVIAr model, identify the root cause of the problems and address them. In the majority of cases this will involve editing the courseware to correct the problem found. We do, however, believe it to be possible to classify the types of courseware problems that will be found and from this infer possible solutions. We investigate this in the future work section, section 10.4.

7.8.3 Case-study

In our case-study the course creator can now validate the databases courseware. The databases Courseware Model is validated against the constraints defined in section 7.6.4, and also the learning context constraints, checking that learning goals and conceptual sequencing constraints are satisfied.

The OCL checker used for the purposes of our case-study is the USE tool (see section 2.4.2 for details). USE provides support for defining the CAVIAr metamodels and then defining runtime snapshots as CAVIAr metamodels instances, CAVIAr models. OCL can be defined on the CAVIAr metamodels and then checked against the CAVIAr models with USE. Figure 7.7 shows a screen-shot of the USE tool, with the CAVIAr metamodels are defined as a UML model. A CAVIAr model is then defined by specifying a system snapshot in USE.



```
Log
compiling specification CAVIAr2_0.use...
done.
Model CAVIAr (27 classes, 33 associations, 1 invariant, 1 operation, 0 pre-/postconditions)
Ready.
```

Figure 7.7: CAVIAr metamodels in the USE tool

In our case-study a violation is identified. The course creator is notified of the constraint violation by the constraint name. In this case the course creator is notified that the “task_has_lecture_LO” has been violated, the OCL checker also indicates the *Topic* that caused the violation.

7.9 Correcting Invalid Courseware

To correct the majority of problems found in courseware the course creator will be required to edit the Courseware Model. Fixes will typically involve:

- The addition or removal of LOs, with certain characteristics.
- Altering the sequencing of courseware topics.
- Altering the structure of the courseware, by adding, removing, splitting or merging topics.
- Editing the Validation Model - false positives are being given by validation engine.

7.9.1 Case-study

The CAVIAr validation violation that was identified in section 7.8.3 was caused by a *Topic* that does not have a LO of type “lecture”. The *Topic* in question is highlighted by the OCL checker. To solve this problem the course creator must locate or create a LO of type “lecture” that satisfies the learning need at the highlighted *Topic* and integrate it into the *topic*. Alternatively, the Validation Model can be edited to allow for *topics* that do not have a LO of type “lecture”. Once the course creator has integrated such a LO into the courseware it can be revalidated.

7.10 Chapter Conclusion

In this chapter we have outlined the activities involved in validating courseware using CAVIAr. These activities make up the CAVIAr courseware validation process. This process empowers the course creator to model constructed courseware and the courseware construction concerns. There are six activities in this process - determining a Domain Model, defining the Learning Context Model, determining a Validation Model, rendering courseware as a CAVIAr model, validating the courseware and then fixing the problems found in the courseware through validation.

We have illustrated the validation process using an example case-study. In our case-study we outlined a typical course module requirements found for a university course. These requirements were used to define CAVIAr models, which were in turn used to validate an example case-study courseware. This example case-study was based on the courseware requirements for DCU course CA218 - “Introduction to Databases”. Validation ensures that courseware satisfies the courseware requirements outlined in the module descriptor.

We have evaluated the courseware validation activities in the validation process through user trials that use the courseware validation activities as part of courseware construction. We will present the details of these user trials in the evaluation chapter, chapter 9.

Chapter 8

CAVIAr Implementation

8.1 Introduction

In the previous three chapters we have described CAVIAr and demonstrated how it can be used to validate courseware. In this chapter we will outline a courseware construction tool based on the CAVIAr that allows for the construction of courseware and its subsequent validation. This tool is known as **M**anagement **I**nfrastructure for **K**nowledge-based **A**daptive **E**-**L**earning (MIKAEL). MIKAEL's purpose is as follows:

- Proof of concept for CAVIAr validation.
- Tool-support for user trial evaluations of CAVIAr.
- Show-case how CAVIAr can be used in courseware construction.

The purpose of this chapter is to discuss a proof of concept application of CAVIAr, which we have developed, known as MIKAEL. MIKAEL is a courseware construction software tool (and Learning Content Management System (LCMS), defined in [Pahl et al., 2007]). The chapter looks at the implementation-specific elements involved in developing tool support based on the CAVIAr metamodels.

The chapter begins by defining a courseware construction process based on the incremental definition of CAVIAr models. MIKAEL has been built to support this courseware construction process. Section 8.3 gives an overview of the MIKAEL software architecture.

MIKAEL has a flexible component-based architecture. In the sections that follow section 8.3 we examine each MIKAEL component in detail by looking at how each component is implemented.

8.2 CAVIAr and Courseware Construction

CAVIAr was designed as a set of models to capture courseware construction concerns, which are then used to validate courseware against a constraints model. CAVIAr can also be used to construct courseware where each of the CAVIAr models represents a courseware construction concern used to derive a skeleton courseware definition. MIKAEL is designed around the application of the CAVIAr models to create courseware. MIKAEL is based on constructing CAVIAr models as part of a courseware construction process. The activities in this courseware construction process can be mapped to the activities in the courseware validation process in chapter 7.

In figure 8.1, we have outlined the courseware construction process using a UML Activity Diagram. In the first activity, the course creator determines a Domain Model that represents the target domain for the courseware being developed. Once a Domain Model is defined, the course creator can then define the learning context for the courseware using the Domain Model. At this point the course creator can also locate and specify learning resources to address the learning need associated with each concept in the Domain Model. A Courseware Model is then generated based on the requirements in the Learning Context Model defined in the preceding activities. The course creator can edit the Courseware Model, until he or she is satisfied with its design. Once courseware construction is complete the course creator can validate the courseware to ensure it satisfies the courseware requirements defined in the learning context and also to ensure that the courseware applies a desired instructional design theory correctly. When the Courseware Model is complete it can be exported into a Technology Enhanced Learning (TEL) specification, such as SCORM or IMS LD (outlined in section 2.2.6).

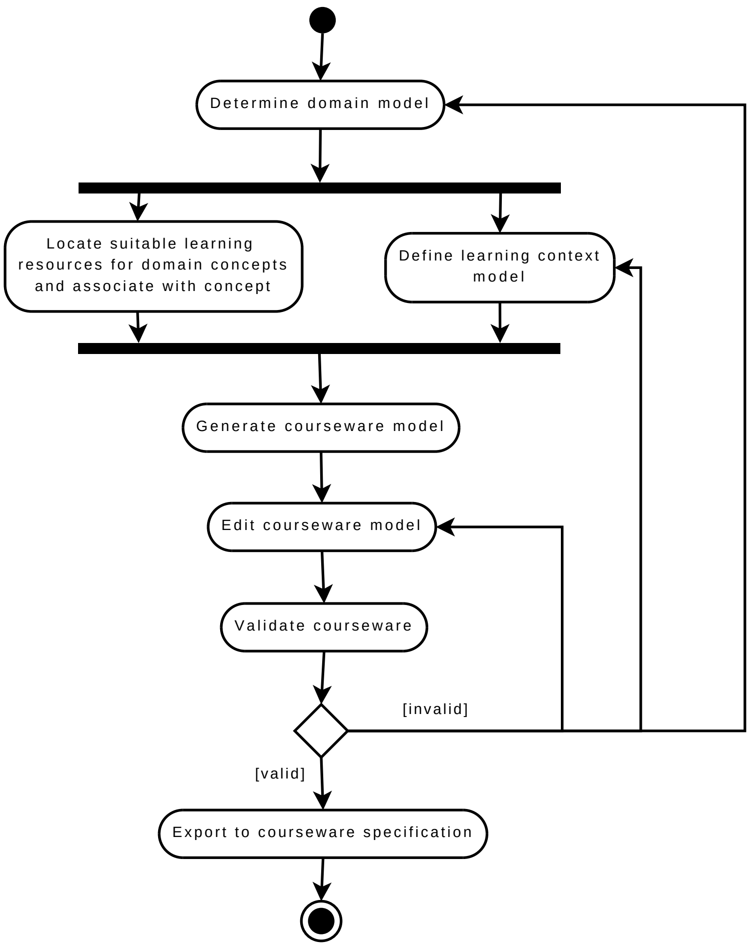

Determine domain model
Locate suitable learning resources for domain concepts and associate with concept
Define learning context model
Generate courseware model
Edit courseware model
Validate courseware
[invalid]
[valid]
Export to courseware specification

Figure 8.1: CAVIAr Courseware Authoring defined in terms of an UML Activity Diagram

8.3 MIKAEL Architecture

The goal of MIKAEL is to support the course creator in constructing courseware through the courseware construction process outlined in figure 8.1. The MIKAEL tool should also support the course creator in the following ways:

- Locate and integrate learning resources.
- Import domain models.
- Export Courseware Model into a TEL specification.
- Validate the Courseware Model.

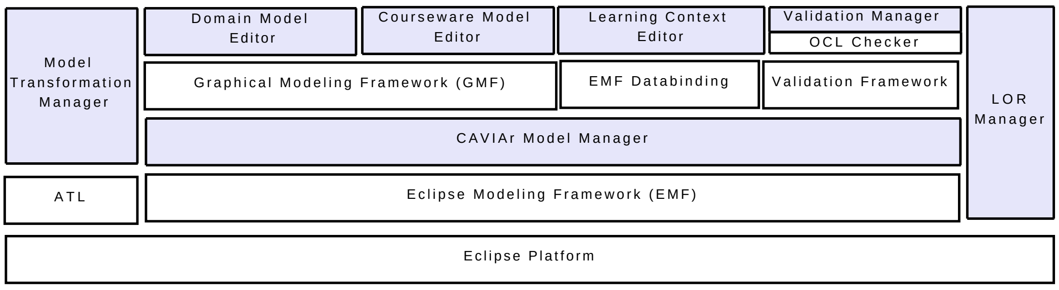


Figure 8.2: Layer view of the MIKAEL Architecture

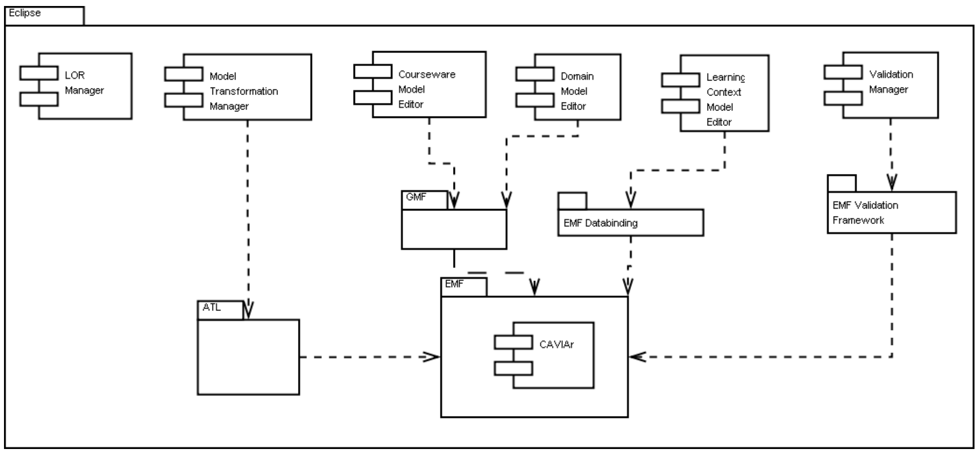


Figure 8.3: Component-based view of the MIKAEL architecture

A high-level architecture of the MIKAEL tool is outlined in figure 8.2. The architecture presented is a layered one, where each layer builds upon the layers below. As outlined in the figure, MIKAEL has be developed using the Eclipse Platform [Arthorne and Laffra, 2004], an open-source Java-based platform. We have highlighted the elements that we have developed in grey. We have also outlined the MIKAEL architecture as a component diagram, using UML, in figure 8.3, to enhance understanding. The interaction between the components we have developed is summarised in figure 8.4. The *Validation Manager/OCL Checker* is half grey and half white as to develop the *Validation Manager* we extended an example Eclipse OCL Checker plug-in. We choose to base MIKAEL on Eclipse as it provides an infrastructure for rapid application development and also provides for various user interface packages to develop an intuitive user interface for the course creator. Also, Eclipse

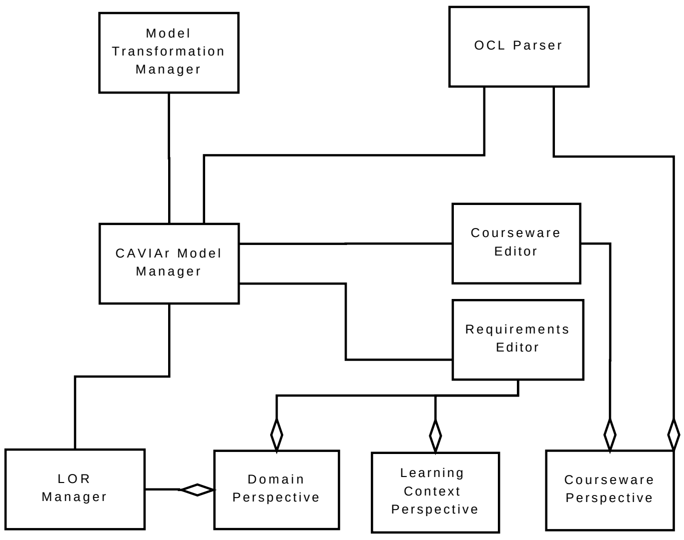


Figure 8.4: MIKAEL Components

has been proven in other courseware construction tool support, such as the Reload LD editor [RELOAD Project, 2005]. The *CAVIAr Model Manager*, which is based on the Eclipse Modelling Framework (EMF) manages the CAVIAr models. MIKAEL provides two intuitive *model-based editors* for constructing CAVIAr models, these are the *Domain Model Editor* and the *Courseware Model Editor* which have been developed using the Graphical Modeling Framework (GMF) [Gronback, 2009].

MIKAEL has been developed to support CAVIAr-based courseware construction. Each component in the MIKAEL architecture is defined to support one or more of the activities outlined in the UML activity diagram in figure 8.1. In the following we list each courseware construction activity and state the MIKAEL component(s) that supports it:

- Determine Domain Model - *Domain Model Editor* and *Transformation Manager*.
- Locate Suitable Learning Resources - *LOR Manager*.
- Define Learning Context Model - *Learning Context Editor*.
- Generate Courseware Model - *Transformation Manager*.
- Edit Courseware Model - *Courseware Model Editor*.

- Validate Courseware Model - *Validation Manager*.
- Export to Courseware Specification - *Transformation Manager*.

Further to this MIKAEL has three Eclipse “perspectives” defined, each of which correspond to a central activity in the courseware construction process in figure 8.1. A perspective is defined as a set of user tools that are needed for some purpose in Eclipse. The defined perspectives are:

- Domain Perspective - Allows for the definition of a subject Domain Model on which the courseware is based, also allows course creator to specify LOs for concepts in Domain Model. Conceptual sequencing constraints can also be defined in the domain perspective.
- Learning Context Perspective - Allows for the definition of learner stereotypes in terms of learning goals and assumed knowledge.
- Courseware Perspective - Allows course creator to define, view and edit the Courseware Model.

Each perspective is defined in detail in section 8.7.

Interoperability in MIKAEL is provided through model transformations, the *Model Transformation Manager* component is responsible for managing model transformations, which leverages the Atlas Transformation Language (ATL) [Jouault and Kurtev, 2005] and associated tools. The MIKAEL *Validation Manager* is based on the EMF Validation Framework [1], which allows for the integration of the Eclipse OCL Checker. In order to locate learning resources the MIKAEL tool must be interoperable with LORs. The *LOR Manager* is responsible for this.

In the following sections we will examine each of the MIKAEL components we have developed in detail. Before this we will outline the Eclipse Framework and the Eclipse tools we have used to develop MIKAEL.

[1]`http://www.eclipse.org/modeling/emf/?project=validation`

8.4 The Eclipse Platform

The Eclipse platform is an open source, highly extensible software platform. Its primary purpose is as a Integrated Development Environment (IDE) for Java development, but other projects have been built on the Eclipse platform which provide support tools to a wide variety of technical and non-technical tasks. The Eclipse platform is built using a plug-in architecture. The Eclipse platform is defined in terms of plug-ins and extensions to the platform are also developed as plug-ins.

The Eclipse *workbench* is the user environment. It is the main window in Eclipse, composed of *editors* and *views*.

The development of Eclipse components is divided into separate projects, for example the Eclipse Tools projects define and integrate different tools into the Eclipse platform, for example in the C/C++ Development Tool (CDT) project IDE functionality for the Eclipse platform is developed for C and C++ developers.

MIKAEL is built on plug-ins from the Eclipse Modelling project. In section 2.3.3 we introduced one such project, EMF. In this section we will outline the other elements of the Eclipse Modeling project that MIKAEL uses.

8.4.1 Graphical Modeling Framework (GMF)

GMF is a generative component for Eclipse, which allows developers to create a graphical modelling user interface for an ECore model. To generate a graphical modelling interface, the GMF developer must define a domain model in ECore, a graphical definition, a tooling definition and a map to integrate them together. GMF is then able to generate the *diagram* code that defines a plug-in to provide the user with a model-based UI for defining instances of the aforementioned ECore domain model.

8.4.2 Model to Model Transformation (M2M)

The M2M project has been set up to integrate a model transformation framework to transform one model type to another model type. There are two main parts to the M2M project, ATL and QVT.

8.4.2.1 Atlas Transformation Language (ATL)

ATL is a model transformation language that allows for model transformation definitions to be defined declaratively and imperatively [Jouault and Kurtev, 2005]. ATL rules define a mapping from source metamodel elements to target metamodel elements. The ATL transformation can then be executed on the source model generating a target model. There are three types of rules in ATL, "matched", "lazy" and "called" rules. Matched rules are invoked when a modelling element which corresponds to the rule's *from* clause is matched in the source model. Lazy rules are called from other rules, but are never executed directly. Lazy rules are normally called with a parameter consisting of a source model artefact. Called rules are similar to lazy rules except a called rule can have any number of parameters. Called rules may also explicitly define a return type or else nothing will be returned.

8.4.2.2 Query/View/Transformation (QVT)

QVT is an OMG specification that allows for model transformations to be defined declaratively and imperatively [OMG, 2005]. Declarative model transformations are defined on two levels - "relations" and "core". Relations are at the specification level defining relationships between MOF model elements. The relations' language supports object pattern matching during transformation execution. Relations can be mapped to a core model, which is defined using a more limited language than that at the relations level. The relations and core express the same semantics but at different levels of abstraction.

There are two Eclipse M2M projects based on QVT, Operational QVT and Declarative QVT [Eclipse Foundation, 2008]. Operational QVT allows the developer to define a QVT transformation relation in the form of a mapping operation. The Declarative QVT is still in development. Its ultimate goal is to provide [Eclipse Foundation, 2008]:

- A dedicated perspective for the Eclipse IDE to define QVT transformations.
- An advanced editor with auto-completion and syntax colouring.

- An execution environment implementation.
- An integrated QVT debugger.

8.4.3 Model to Text Transformation (M2T)

The M2T project allows for the generation of text from models [Eclipse Foundation, 2009]. There are five components of the M2T project:

- Java Emitter Template (JET) - Provides a code generation facility using JSP like template files, where developers write templates for the code to be generated [Popma, 2003]. JET not only allows the generation of Java, but also structured languages such as SQL and XML.
- Acceleo - An implementation of the OMG Model Transformation Language (MTL) language specification.
- Xpand - A statically-typed template language.
- M2T Core - Invocation framework allowing clients to invoke model-to-text solutions independently of a model-to-text language.
- M2T shared - Consists of infrastructure elements that can be shared between model-to-text languages.

8.4.4 Model Development Tools (MDT)

The aim of the MDT project is as follows [Eclipse MDT, 2008]:

- To provide an implementation of standardised metamodels.
- To provide exemplary tools for developing models based on those metamodels.

Within the MDT project is the OCL component. This is used in MIKAEL for defining the CAVIAr Validation Model. The OCL component provides the following support for OCL integration into the Eclipse platform [Eclipse MDT, 2008]:

- APIs for parsing and evaluating OCL constraints and queries on EMF models.
- An Ecore implementation of the OCL abstract syntax model.
- A visitor API for analysing and transforming the Abstract Syntax Tree (AST) of OCL expressions.
- An extensibility API for clients to customise the parsing and evaluation environments used by the parser.

8.5 CAVIAr Model Manager

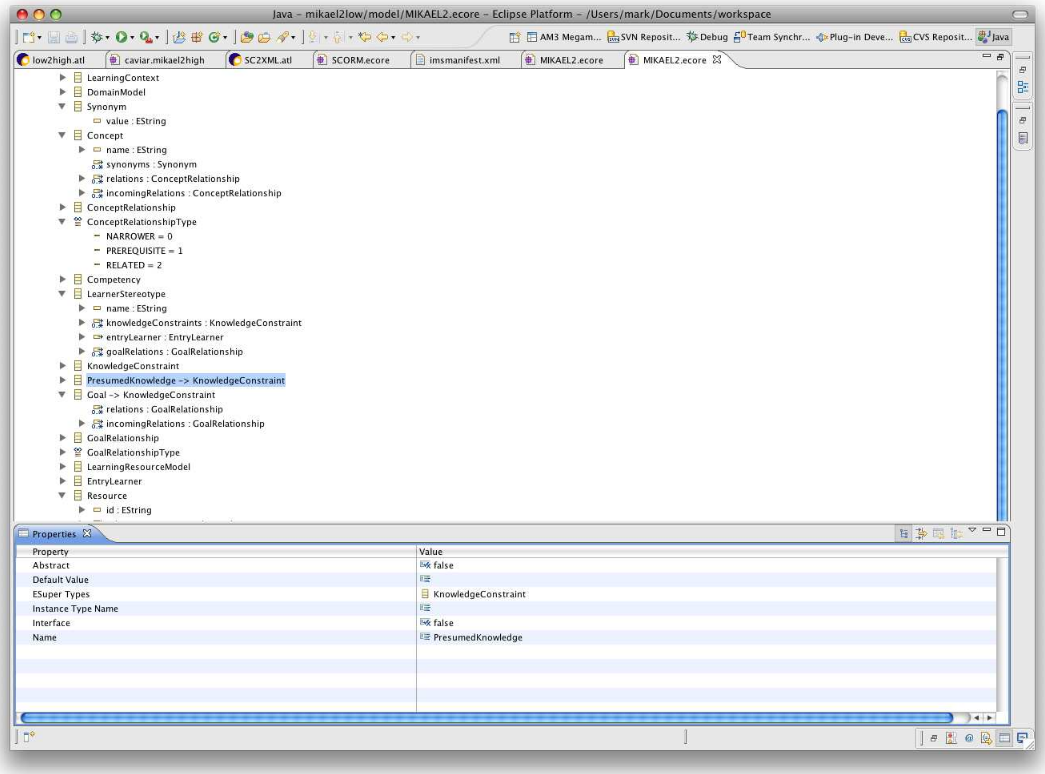
The CAVIAr-based approach to courseware construction is a systematic approach to the development of CAVIAr models for courseware construction. The management of CAVIAr models, in MIKAEL, is the responsibility of the CAVIAr Model Manager. The CAVIAr Model Manager is built on the Eclipse Modelling Framework (EMF). The CAVIAr Model Manager manages the updating of CAVIAr models created through the EMF API. The MIKAEL tool is based on the definition of CAVIAr in EMF. The CAVIAr metamodels are defined in ECore in the CAVIAr Model Manager.

Figure 8.5 shows the manipulation of the CAVIAr ECore metamodel using the EMF editor. An ECore model is defined in EMF using the default tree-based user interface.

8.6 Model-based Editors

The central components of the MIKAEL tool are the model editors developed using the Eclipse Graphical Modeling Framework (GMF). The GMF editors provide the course creator with a model-based user interface for courseware construction.

MIKAEL has two GMF editors, the *Domain Model Editor* and the *Courseware Model Editor*. The *Domain Model Editor* allows the course creator to edit the CAVIAr Domain Model and also allows for some elements of the learning context to be defined. The *Courseware Model Editor* allows the course creator to edit the Courseware Model.



```
▶ LearningContext
▶ DomainModel
▼ Synonym
      value : EString
▼ Concept
   ▶ name : EString
      synonyms : Synonym
   ▶ relations : ConceptRelationship
   ▶ incomingRelations : ConceptRelationship
▶ ConceptRelationship
▼ ConceptRelationshipType
      NARROWER = 0
      PREREQUISITE = 1
      RELATED = 2
▶ Competency
▼ LearnerStereotype
   ▶ name : EString
   ▶ knowledgeConstraints : KnowledgeConstraint
   ▶ entryLearner : EntryLearner
   ▶ goalRelations : GoalRelationship
▶ KnowledgeConstraint
▶ PresumedKnowledge -> KnowledgeConstraint
▼ Goal -> KnowledgeConstraint
      relations : GoalRelationship
   ▶ incomingRelations : GoalRelationship
▶ GoalRelationship
▶ GoalRelationshipType
▶ LearningResourceModel
▶ EntryLearner
▼ Resource
   ▶ id : EString
```

Property	Value
Abstract	false
Default Value	
ESuper Types	KnowledgeConstraint
Instance Type Name	
Interface	false
Name	PresumedKnowledge

Figure 8.5: Defining the CAVIAr metamodels in ECore using Eclipse

A GMF editor is not provided for the Learning Resource Model and the Learning Context Model in MIKAEL. A model-based user interface would not be an optimal method of displaying this data due to its verbose nature. Instead the information in these models is portrayed to the course creator using table and list based views. We have outlined how these views are used in Eclipse perspectives for the Learning Resource Model in section 8.7.1 and for the Learning Context Model in section 8.7.2.

8.6.1 Domain Model Editor

The *Domain Model Editor* is principally concerned with the CAVIAr Domain Model. The *Domain Model Editor* allows the course creator to create a Domain Model through an intuitive user interface.

Figure 8.6 shows a screen-shot of the *Domain Model Editor*. As shown, the environment that the course creator defines the Domain Model in is comparable to other cognitive tools used in the educational domain such as mind-maps [Budd, 2004]. There are two types of nodes that can be added and used in the *Domain Model Editor*, *concept* nodes and *synonym* nodes. Three relationship types between concepts are available, *NARROWER*, *RELATED* and *PREREQUISITE*. A relationship is also defined to relate a concept to its synonym.

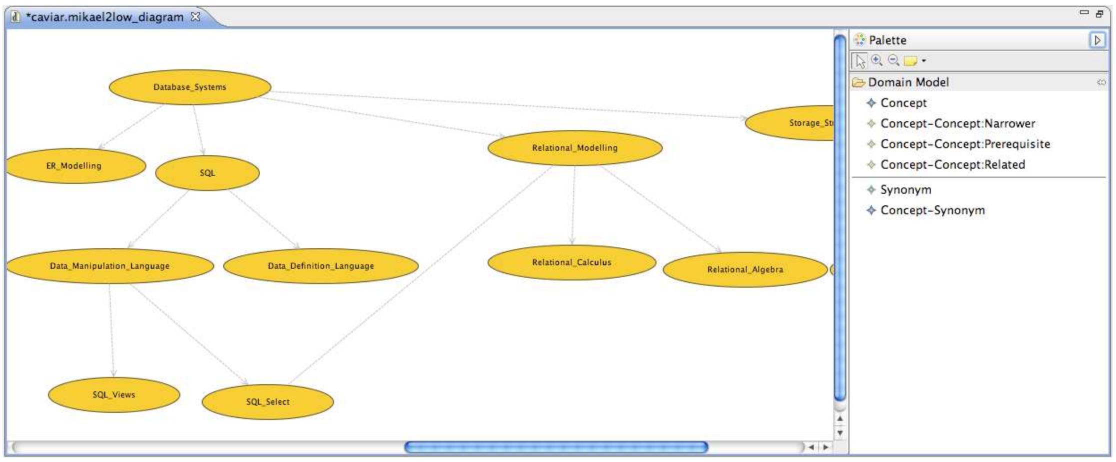


Figure 8.6: Screen-shot of domain model view

8.6.2 Courseware Model Editor

The *Courseware Model Editor* is principally concerned with the Courseware Model in CAVIAr. The *Courseware Model Editor* allows the course creator to create a Courseware Model through an intuitive interface.

The editor has two node types, *topic* nodes and *entryLearner* nodes. There are two relationship types between *topic* nodes, *PART_OF* and *SEQUENCED_AFTER*. There is also a relationship to relate the graphical representation of an entry learner condition to a topic.

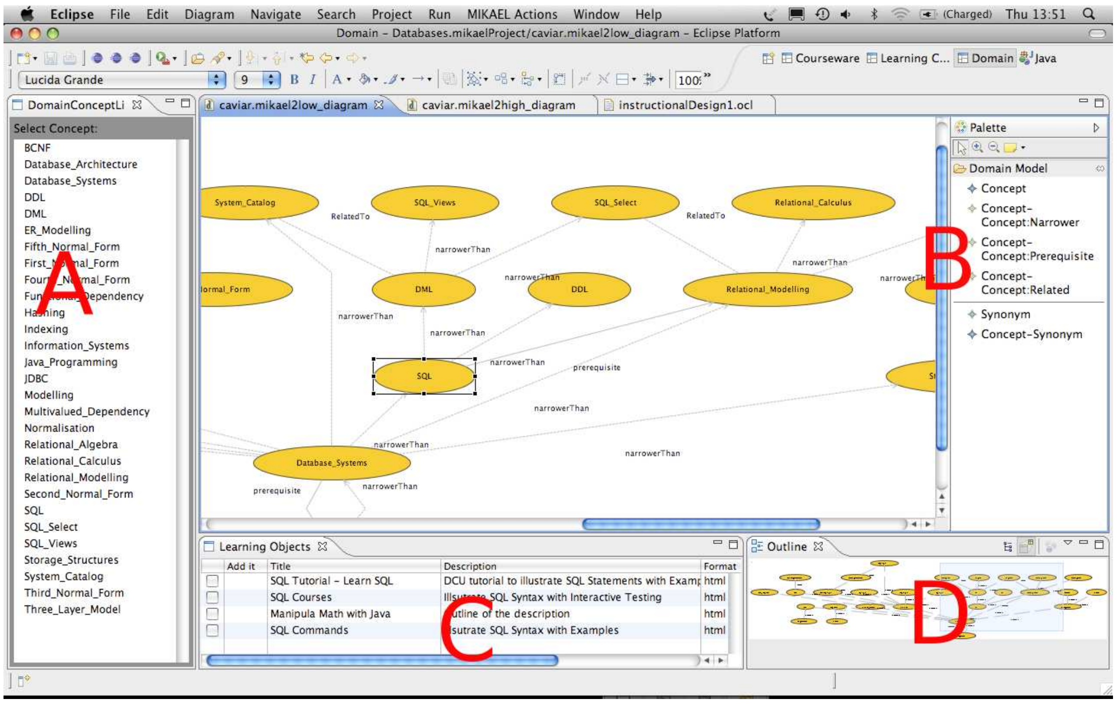
8.7 MIKAEL Perspectives

The Eclipse framework allows for the definition of perspectives. A perspective is a defined set of editors and views. In this section we will examine the perspectives designed for MIKAEL. Each perspective supports one or more activities in the courseware construction process illustrated in figure 8.1 and can also be used to support the courseware validation process activities illustrated in figure 7.1. In the following we outline each perspective and the activities it supports in the courseware validation and courseware construction processes. These are as follows:

- Domain Perspective - For defining and editing a domain model definition.
  - Courseware Validation Process - Determine Domain Model.
  - Courseware Construction Process - Determine Domain Model, locate learning resources.
- Learning Context Perspective - For defining learning context information in the context of the domain information.
  - Courseware Validation Process - Determine Learning Context Model.
  - Courseware Construction Process - Define Learning Context Model, generate Courseware Model.
- Courseware Perspective - Allows for the development of a Courseware Model.
  - Courseware Validation Process - Import courseware, determine Validation Model, validate courseware, edit courseware.
  - Courseware Construction Process - Edit Courseware Model, validate Courseware Model, export to courseware specification.

8.7.1 Domain Perspective

The domain perspective is made up of the *Domain Model Editor*, a Domain Model Outline, Concept List View and the Learning Object Repository (LOR) View. The domain perspective allows the course creator to manipulate and create a Domain Model to be used in courseware construction. The *Domain Model Editor* gives the course creator a graphical model-based environment in which to define a Domain Model, as outlined in section 8.6.1. As Domain Models used in courseware construction can be quite extensive and difficult to view, MIKAEL provides the course creator with two ways to manage the Domain Model at the macro level - a Domain Model outline and concept list view. The Domain Model outline provides a graphical overview of the Domain Model view. The concept list view simply lists all the concepts in the Domain Model, in alphabetical order, providing the course creator with a simple sorted list to find a concept. When the course creator clicks a concept in the concept list it is highlighted in the *Domain Model Editor*.



Select Concept:
BCNF
Database_Architecture
Database_Systems
DDL
DML
ER_Modelling
Fifth_Normal_Form
First_Normal_Form
Fourth_Normal_Form
Functional_Dependency
Hashing
Indexing
Information_Systems
Java_Programming
JDBC
Modelling
Multivalued_Dependency
Normalisation
Relational_Algebra
Relational_Calculus
Relational_Modelling
Second_Normal_Form
SQL
SQL_Select
SQL_Views
Storage_Structures
System_Catalog
Third_Normal_Form
Three_Layer_Model

Learning Objects

	Add it	Title	Description	Format
☐		SQL Tutorial – Learn SQL	DCU tutorial to illustrate SQL Statements with Examp	html
☐		SQL Courses	Illsutrate SQL Syntax with Interactive Testing	html
☐		Manipula Math with Java	utline of the description	html
☐		SQL Commands	lsutrate SQL Syntax with Examples	html

Figure 8.7: Screen-shot of domain perspective

Each concept in the Domain Model represents a potential learning need. To overcome this learning need, the course creator must assign LOs to that concept in the Domain Model, where the LO provides for learning on the concept it is assigned to. To this end the LOR View is integrated into the domain perspective. This view allows for the automated search for LOs that satisfy the learning need represented by the selected concept in the *Domain Model Editor*.

The screen-shot in figure 8.7 outlines the domain perspective. The figure is annotated A to D, indicating the different parts of the perspective as follows:

- A - Concept List View.
- B - *Domain Model Editor* - graphical model on the left, tool palette on the right.
- C - LOR View - outlining the LOs associated with the selected concept. The metadata of the LO is also outlined in the view.
- D - Domain Model Outline.

8.7.2 Learning Context Perspective

The learning context perspective is used for defining the CAVIAr learning context as outlined in section 5.4. The perspective is centred around the definition of learner stereotypes, where a learner stereotype is defined in terms of conceptual presumed knowledge and conceptual learning goals.

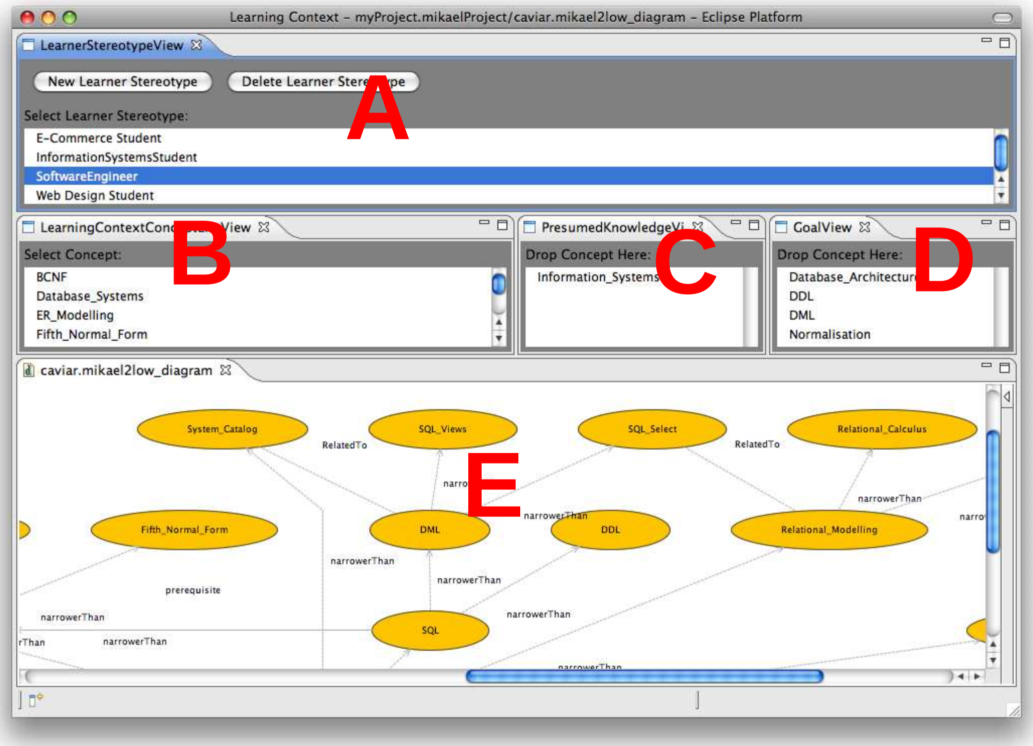


Figure 8.8: Screen-shot of learning context perspective

Figure 8.8 is a screen-shot of the learning context perspective. The main controlling element of this perspective is the learner stereotype view annotated with “A”. Using this view the course creator can add new learner stereotypes and delete existing ones. When a learner stereotype is selected the views below it allow for the definition of the Learning Context Model as follows:

- A - Learner Stereotype List View.
- B - Concept List View - lists the concepts in the Domain Model.
- C - Presumed Knowledge View - drag a concept from *concept list view* (B) to here for that concept to be presumed knowledge for the selected stereotype.
- D - Goal View - drag a concept from the *concept list view* (B) to here for that concept to be a goal for the selected stereotype.
- E - Domain Model View - graphical view of the concepts in Domain Model, allows the course creator to easily see the context of a given concept.

We have outlined the activities involved in creating a learner stereotype in MIKAEL in a UML Activity Diagram, figure 8.9.

8.7.3 Courseware Perspective

The courseware perspective allows the course creator to edit and view the CAVIAr Courseware Model. This is done using the *Courseware Model Editor* and the entry learner view. As outlined in section 8.6.2, the courseware model editor allows the course creator to edit and view topics and topic relations in a Courseware Model. Using the courseware perspective the course creator can define entry learner conditions on courseware topics, using a separate *Entry Learner View*. When an entry learner condition is selected in the Courseware Model editor, the *Entry Learner View* displays the details of the entry learner condition and allows the course creator to edit the entry learner definition.

In figure 8.10 we have annotated a screen-shot of the courseware perspective, as follows:

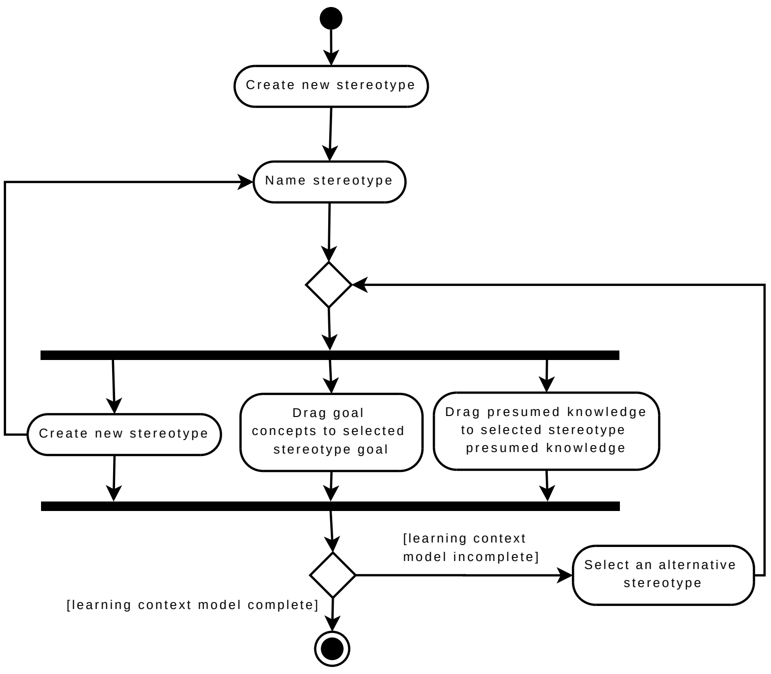
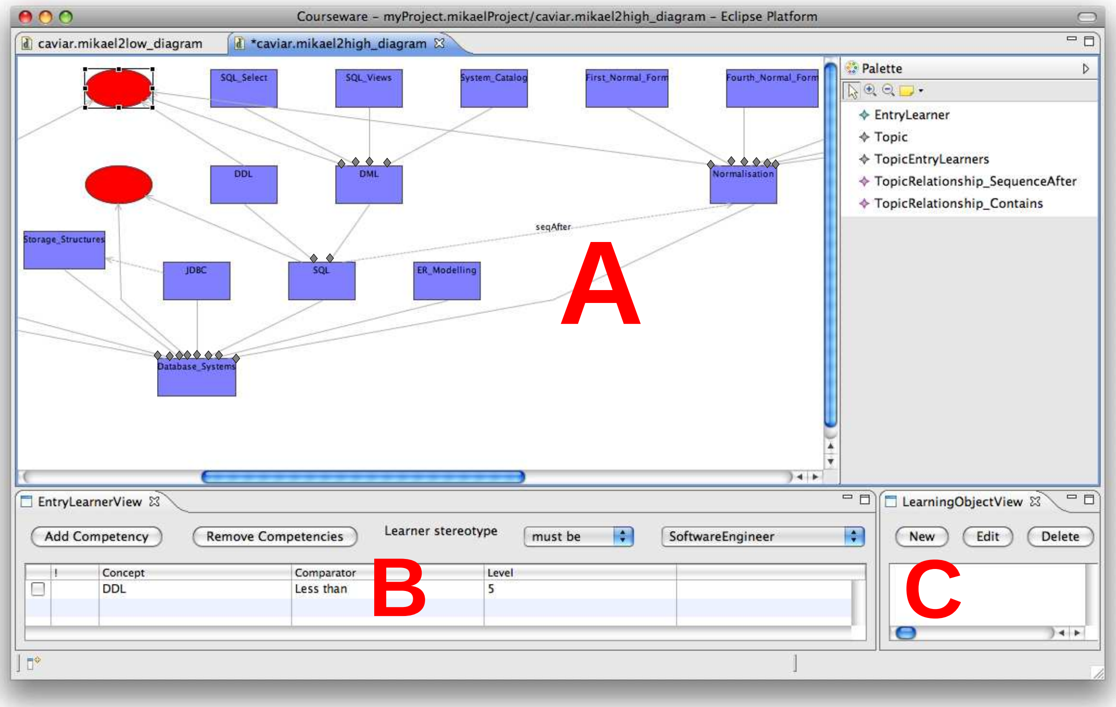

Create new stereotype
Name stereotype
Create new stereotype
Drag goal concepts to selected stereotype goal
Drag presumed knowledge to selected stereotype presumed knowledge
[learning context model incomplete]
Select an alternative stereotype
[learning context model complete]

Figure 8.9: UML Activity Diagram outlining tasks involved in defining learning context

- A - *Courseware Model Editor* - On the left is a graphical model of the courseware, while on the right is the tool palette for editing the Courseware Model.
- B - *Entry Learner View* - There are two main parts of the *Entry Learner View*. The top of the view indicates the learner stereotype that the selected entry learner condition affects. The lower part of the view lists the knowledge elements and comparators associated with this entry learner condition. Buttons above this list allow the course creator to add and remove knowledge elements to the entry learner condition through the add and remove competency buttons.
- C - Learning Object Editor - This editor is activated when the course creator clicks on a topic in the Courseware Model Editor. The view lists the LOs that are referenced by the selected topic. The learning object editor can also be used to add LOs to topics and edit the existing LOs associated with courseware topics.



Learner stereotype: must be SoftwareEngineer

!	Concept	Comparator	Level
☐	DDL	Less than	5

Figure 8.10: Screen-shot of the courseware perspective

8.8 Model Transformation Manager

Model transformation technology is an integral part of MIKAEL. Model transformation technology is used at any point in the course construction process that requires the creation of a new model. These are:

- Importing a Domain Model - Using model transformation technology for importing data, allows MIKAEL to be flexible enough to be able to import domain models from any source as long as a metamodel is defined for the source data and a transformation from the source metamodel to the CAVIAr Domain Model's metamodel can be defined.

- Importing Courseware Model for Validation - the MIKAEL tool can be used to import a courseware definition, which was developed using some other courseware construction tool, for CAVIAr-based validation.

- Courseware Model Generation - model transformations are used in MIKAEL to define how a Courseware Model is generated based on the courseware construction concerns defined in the CAVIAr Learning Context Model, as outlined in figure 8.1.

- Exporting Courseware - MIKAEL can export courseware constructed into any courseware specification as long as a metamodel is defined for the target data type and a transformation from CAVIAr Courseware Model's metamodel to the target metamodel can be defined.

When developing the Model Transformation Manager, we looked to base it on a transformation framework that was:

- Stable.

- Well established.

- Well supported.

- Allowed for both declarative and imperative transformation definitions - some of the CAVIAr transformation definitions could not be defined declaratively.

- Could be used with EMF models.

A range of transformation languages were evaluated based on the work of Czarnecki and Helsen, who surveyed model transformation approaches based on their features [Czarnecki and Helson, 2006].

We decided to build the Model Transformation Manager on the ATLAS Transformation Language (ATL) framework. ATL is a model transformation specification language that uses declarative and imperative constructs. Declarative programming is the preferred method of defining model transformations by expressing mappings from one metamodel to another as it is a direct mapping between model elements and it is intuitive. Imperative constructs allow for the definition of transformation specifications that are difficult to express declaratively [ATLAS Group, 2006]. The language allows the user to define mappings from source metamodel constructs to target metamodel constructs [Bézivin et al., 2005]. ATL is a well-established and stable model transformation language, which can be used with EMF models. It also well supported by an active user group.

The MIKAEL Model Transformation Manager can also allow for data sources that are not defined in the modelling technical space. These data sources can be converted as long as an XML binding is available for the data source. The Model Transformation Manager uses the ATLAS MegaModel Management Tool (AM3) for this. The AM3 tool (AM3 XML injector) can load in an XML source and create an XML model for the XML data source. The XML model can then be transformed to a domain specific model.

8.8.1 Importing a Domain Model

The first task in the courseware construction process, illustrated in figure 8.1, is to load a domain model. To do this, the course creator will generally source a domain model from some third party.

An ontology provided by some trusted third party can be used to define a CAVIAr Domain Model, or at least to provide a starting point. The Semantic Web makes use of ontologies to provide a common reference point for syntax used in web applications. With the growing importance of Semantic Web applications, there is an increasing number of ontologies available [Hendler, 2008].

Semantic Web ontologies have an XML binding. In order for these ontologies to be used the ontology definition must be mapped from the XML technical space to the modelling technical space resulting in the generation of a CAVIAr Domain Model. To convert XML to an XML model, the AM3 XML injector is used in MIKAEL. When an XML model has been generated, the XML model can be transformed into a CAVIAr Domain Model using ATL. In figure 8.11, we outline the steps involved in generating a CAVIAr Domain Model from an ontology with an XML binding.

Examples of ontology and modelling languages that can be transformed into a Domain Model are as follows:

- Web Ontology Language (OWL) [W3C, 2004] defined ontologies - This is the main ontology definition language used on the Semantic Web. OWL is based on description logics and has XML syntax.

- XML Topic Maps (XTM) [Pepper and Moore, 2001] - An XML based approach to create indexes, where topics are related to other topics.
- XML Schemata - An XML vocabulary definition.
- UML2 models - OMG defined software engineering modelling specification.

We exemplify the transformation in figure 8.11 by outlining how a SKOS ontology is transformed to a CAVIAr Domain Model as follows:

1. Convert SKOS from an XML serialisation to XML ECore model that conforms to the XML metamodel, using AM3.
2. Transform the XML model into a CAVIAr Domain Model using ATL.

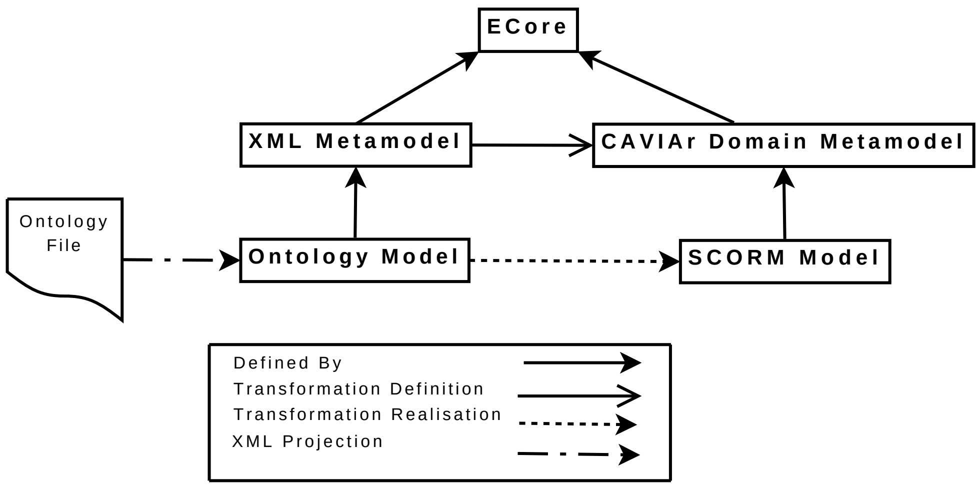


Figure 8.11: Transformation of ontology XML file to CAVIAr Domain Model

As outlined the SKOS XML is loaded into an XML model using the AM3 XML injector. The XML model is then transformed to a CAVIAr Domain Model ECore representation using ATL. The transformation definition can be found in appendix B. Here, we highlight some of the key transformation rules as follows:

- XML elements, named “skos:concept” are transformed into a CAVIAr Domain Model *concept* - *concept2concept* rule.
- XML elements named “skos:broader” are transformed into CAVIAr *narrower* relationship between CAVIAr concepts - *SKOSRelationshipBroader2conceptRelationship* rule.
- XML elements named “skos:related” are transformed into a CAVIAr *related* relationship between CAVIAr concepts - *SKOSRelationshipRelated2conceptRelationship* rule.

8.8.2 Importing a Courseware Model for Validation

The MIKAEL tool allows the course creator to validate courseware using CAVIAr validation, covered in section 8.9. The course creator can also use MIKAEL to validate courseware that has not been constructed using our tool. To allow for this, MIKAEL provides the course creator with the facility to import courseware.

Specification compliant courseware, such as SCORM courseware, can easily be imported into MIKAEL by mapping the courseware specification metamodel to the CAVIAr Courseware Model’s metamodel. We outline these mappings in more detail in section 8.8.4. More challenging is the importation of non-standard courseware, such as Adaptive Educational Hypermedia (AEH). In this section, we describe how to import one type of AEH, LAOS-based AEH (see section 3.2.2.1 for details on LAOS), into MIKAEL for the purposes of validation. More details on how we achieved interoperability between LAOS and CAVIAr can be found in [Melia and Pahl, 2008].

To validate AEH defined by MOT (outlined in section 3.2.2.1), using CAVIAr, the LAOS models must be transformed into CAVIAr models. To do this, a metamodel must exist for LAOS and a transformation definition must be defined from the LAOS metamodel to the CAVIAr metamodels. We define two metamodels for LAOS, one for LAOS’s static elements, based on the Common Adaptation Framework (CAF), and one for the AEH adaptive rules in LAOS, based on LAG.

A CAF ECore metamodel can be defined based on the CAF XML definition, defined as an XML Document Type Definition (DTD) [Cristea et al., 2007]. This DTD can be converted to an XML schema using an XML editing tool, such as XMLSpy [Altova, 2005]. This XML schema is then used to generate the CAF ECore metamodel [Steinburg et al., 2008]. Due to ECore's superior expressivity, additional semantics can be added to the ECore model definition that are implicit in the DTD, as follows:

- An explicit link between *Link* and *Attribute* can be defined as only an implicit link exists in the CAF XML DTD.
- A "value" attribute must be added to CAF elements that contain text as a means to represent the text.
- Specify ordered relationships for elements that have an implicit ordering in the CAF DTD.

The final CAF metamodel is illustrated in UML in figure 8.12.

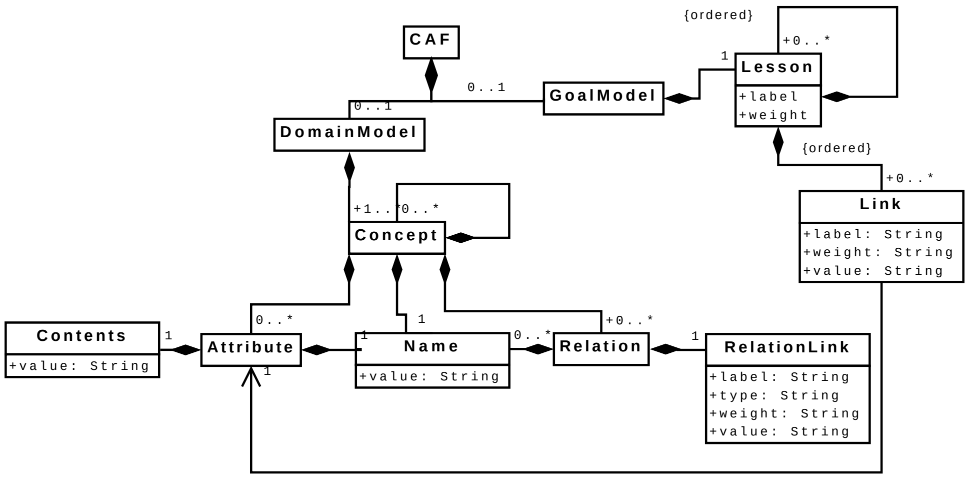


Figure 8.12: CAF metamodel defined using ECore

Once the CAF ECore metamodel has been defined, a transformation mapping from the CAF metamodel to the CAVIAr metamodels can be defined. These mapping define how each modelling construct in the CAF metamodel maps to a CAVIAr metamodel construct.

We provide details on the mappings we have used in our interoperability experiments in [Melia and Pahl, 2008].

LAG rules are used to define adaptive behaviour in LAOS. In the CAVIAr Courseware Model adaptivity is defined by specifying restrictions on the sequencing of courseware topics and/or restrictions on learner stereotypes that can access a given topic. Adaptivity in the CAVIAr Courseware Model is defined using modelling constructs, such as defining a sequencing constraint relationship between topics. To transform the adaptive rules defined using LAG to CAVIAr Courseware Model constructs the LAG language must be defined in the modelling technical space. We have defined a limited metamodel for the LAG abstract syntax in figure 8.13. This metamodel allows for the representation of LAG in the modelling space. A LAG rule can then be parsed to create a LAG model. A transformation is defined mapping the LAG metamodel to the CAVIAr Courseware Model's metamodel which is integrated with CAVIAr models generated from the previous CAF transformation.

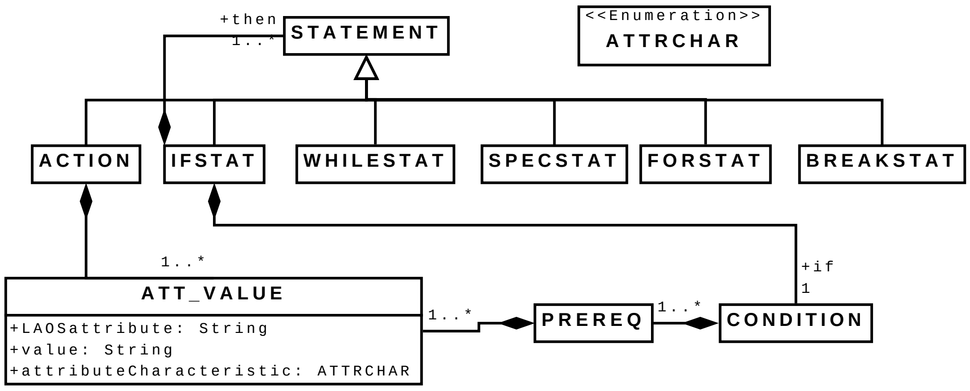


Figure 8.13: LAG defined as ECore metamodel

A LAG "sequencing" rule specifies that when a particular part of the LAOS Domain Model is accessed, a different part of the AEH is to be made available to the learner. An example of a LAG sequencing rule is outlined in listing 8.1. This type of LAG rule is made up of two different parts, a condition and an action. The condition checks if the attribute "title" of concepts in the domain model has been accessed - "access" being the LAG attribute. In turn, the action sets the LAOS "text" attribute to be shown - "show" being the attribute being set.

Listing 8.1: LAG sequencing rule

```
IF (DM.Concept.title.access == 'true') THEN
   (DM.Concept.text.show == 'true')
```

This rule is parsed by the Model Transformation Manager and creates an instance of the the LAG metamodel - a LAG model, illustrated in figure 8.14.

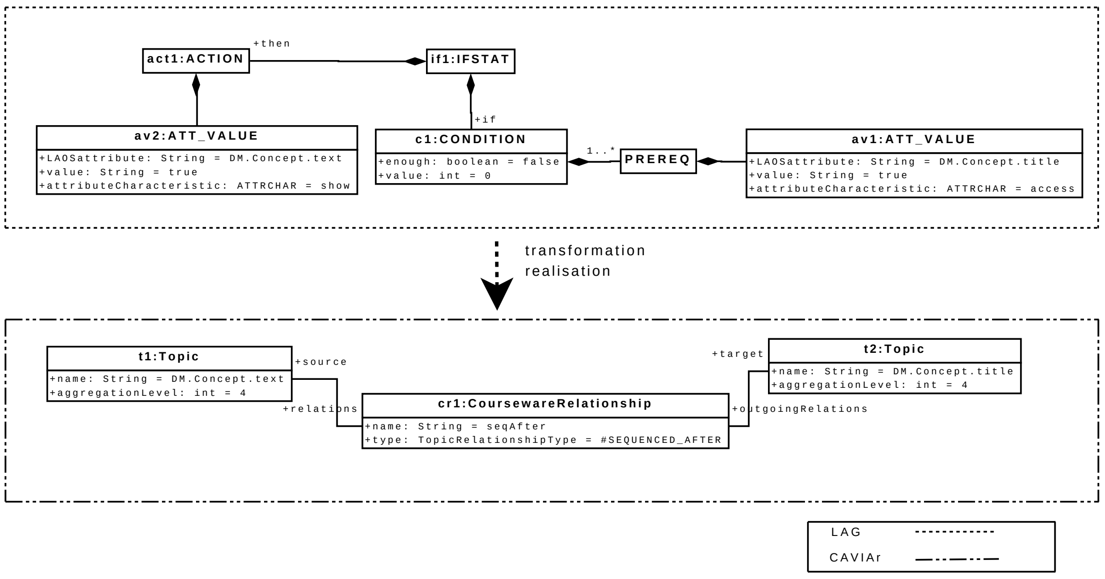

+then
act1:ACTION
if1:IFSTAT
+if
av2:ATT_VALUE
+LAOSattribute: String = DM.Concept.text
+value: String = true
+attributeCharacteristic: ATTRCHAR = show
c1:CONDITION
+enough: boolean = false
+value: int = 0
1..*
PREREQ
av1:ATT_VALUE
+LAOSattribute: String = DM.Concept.title
+value: String = true
+attributeCharacteristic: ATTRCHAR = access
transformation
realisation
t1:Topic
+name: String = DM.Concept.text
+aggregationLevel: int = 4
+source
+relations
cr1:CoursewareRelationship
+name: String = seqAfter
+type: TopicRelationshipType = #SEQUENCED_AFTER
+outgoingRelations
+target
t2:Topic
+name: String = DM.Concept.title
+aggregationLevel: int = 4
LAG
CAVIAr

Figure 8.14: Transformation of LAG model to CAVIAr Courseware Model

When a LAG model has been constructed for the rule in listing 8.1, the LAG rule model can be transformed into a CAVIAr Courseware Model. To do this a transformation mapping from the LAG metamodel to the CAVIAr metamodels is defined. This LAG rule states when *DM.Concept.title* attribute is accessed show the *DM.Concept.text attribute*. The transformation maps this type of LAG rule to a CAVIAr Courseware Model where each attribute in the LAG condition and action, is a courseware topic. The topic mapped to the "title" attribute in the LAG rule is the source topic of a *SEQUENCED_AFTER* courseware relationship. The target topic of the *SEQUENCED_AFTER* rule is the topic mapped to the "text" attribute. We demonstrate how this transformation would take effect through the example transformation in figure 8.14.

8.8.3 Courseware Model Generation

Once the Learning Context Model has been defined, MIKAEL can be used to generate the Courseware Model. Again, here MIKAEL uses ATL to generate a Courseware Model from the Learning Context Model. For this ATL transformation the Learning Context Model is taken in as the input model to the transformation, and a CAVIAr Courseware Model is the output.

The transformation definition is in effect a definition of the instructional design to use in the courseware, defining how the courseware construction concerns map to a courseware definition. Different transformation definitions result in differing Courseware Models. For example one transformation could define a courseware that starts with topics covering more specialised concepts in the Domain Model and works towards topics covering broader concepts, another might do the opposite covering the topics that address broader concepts first.

An example of a transformation that takes the Learning Context Model and generates a Courseware Model can be found in Appendix B. This transformation maps concepts that are goal concepts for any learner stereotypes in the Learning Context Model to topics in the Courseware Model. EntryLearner conditions are placed on topics so that each learner stereotype is delivered only topics associated with that stereotype's goal. Conceptual sequencing constraints are mapped to topic sequencing relationships, specifying that one topic must be delivered before another.

8.8.4 Exporting a Courseware Model

The MIKAEL tool supports the export of a CAVIAr Courseware Model into a TEL specification, such as SCORM or IMS LD. This allows the courseware to be deployed using one of the mainstream LMSs such as Moodle[2] or Blackboard[3].

In figure 8.15, we outline the transformations that creates a SCORM manifest file from a CAVIAr Courseware Model. As outlined the Courseware Model is transformed into a

[2]`http://moodle.org`
[3]`http://www.blackboard.com`

SCORM model, which is an instance of a predefined SCORM metamodel. The aim here is to generate a SCORM manifest file, which is an XML file. The SCORM model must therefore be converted to an XML file. To do this the SCORM model is transformed to an XML model that conforms to the XML metamodel. AM3 XML injector is used to generate an XML document from the XML model.

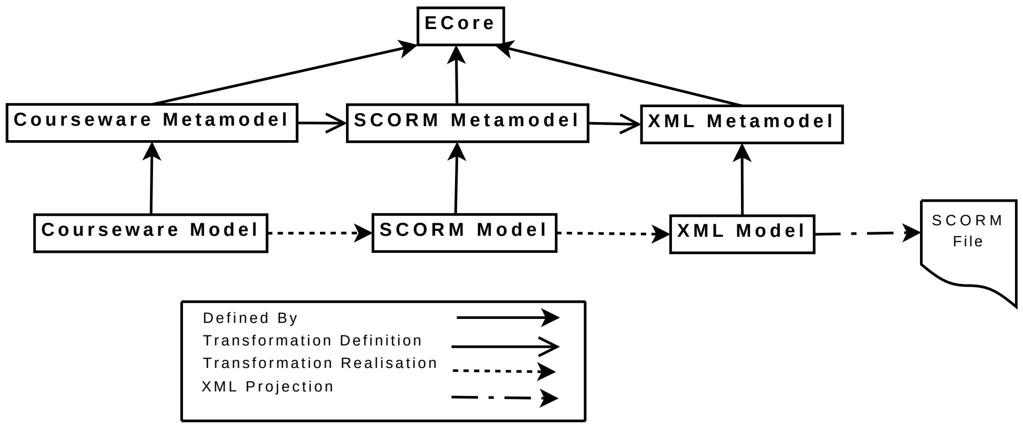


Figure 8.15: Transformation from Courseware Model to SCORM XML binding definition

The ATL to transform a CAVIAr Courseware Model into a SCORM XML manifest can be found in appendix B.

8.9 Validation Manager

EMF has a model validation framework[4] that allows for the definition of constraints on EMF meta-models. These constraints are then checked on the corresponding EMF models constructed. The EMF validation framework allows for constraints to be defined in Java and the Object Constraint Language (OCL). We extended the validation service to check that the Courseware Model adheres to OCL constraints defined in the CAVIAr Validation Model.

As described in chapter 6, the CAVIAr Validation Model is defined using OCL. The definition of constraints on the CAVIAr Courseware Model's metamodel enables the defini-

[4]`http://www.eclipse.org/modeling/emf/?project=validation`

tion of what is structurally incorrect for a given CAVIAr Courseware Model. To validate a CAVIAr Courseware Model the OCL constraints are checked against the CAVIAr Courseware Model defined by the course creator.

To provide for validation support of the OCL constraints defined in the Validation Model, the EMF validation framework is extended to accept an OCL file, defined as the CAVIAr Validation Model, to be checked against a CAVIAr model defined by the course creator. An OCL constraint provider is defined and integrated into the EMF validation service, using the *constraintProviders* extension point, where an extension point is a mechanism provided by the Eclipse Platform to extend its functionality.

The OCL constraint handling functionality is handled as a plug-in in Eclipse. In order to make Eclipse aware of the functionality provided by the plug-in, it must be described in a file called *plugin.xml*. In listing 8.2 we have provided the extension point definition for the *OCLConstraintProvider*, which defines the *OCLConstraintProvider* as the constraint handler for packages with the URI of *www.caviar.dcu.ie* - this being the URI identifying a CAVIAR metamodel.

Listing 8.2: Constraint provider extension point for OCLConstraintProvider

```
<extension
         point="org.eclipse.emf.validation.constraintProviders"
         id="oclProvider">
      <!-- Custom constraint provider using OCL documents -->
      <constraintProvider
            class="org.eclipse.emf.validation.examples.ocl.OCLConstraintProvider"
            category="Constraints␣from␣an␣OCL␣Document"
            cache="false">
         <package namespaceUri="www.caviar.dcu.ie"/>
         <ocl path="constraints/validationModel1.ocl"/>
      </constraintProvider>
   </extension>
```

For full details on the *plugin.xml* defining the plug-in to handle OCL constraints definitions, please consult appendix B.

8.10 Learning Object Repository (LOR) Manager

To be useful, MIKAEL must provide the course creator with an easy search mechanism for LOs. MIKAEL can interoperate with Learning Object Repositories (LORs) and automatically search for LOs that could be used in a given courseware. The *LOR Manager* is responsible for managing connections to LORs, querying LORs, and interpreting the results returned from LORs.

The LOR manager uses two main strategies for searching LORs:

- A screen-scraping approach [Hemenway and Calishain, 2003].
- A service-based query approach [Simon et al., 2005].

8.10.1 Screen-Scraping LOR Queries

The screen-scraping approach is carried out by defining a LO query as a URL, and passing the URL to the LOR of choice. Parameters such as the keywords to use in the search are passed in the URL. The result is returned as HTML. This HTML is parsed by a custom parser class that is aware of the structure of the HTML returned by the LOR. The parser then generates a CAVIAr *metadata* model from the result.

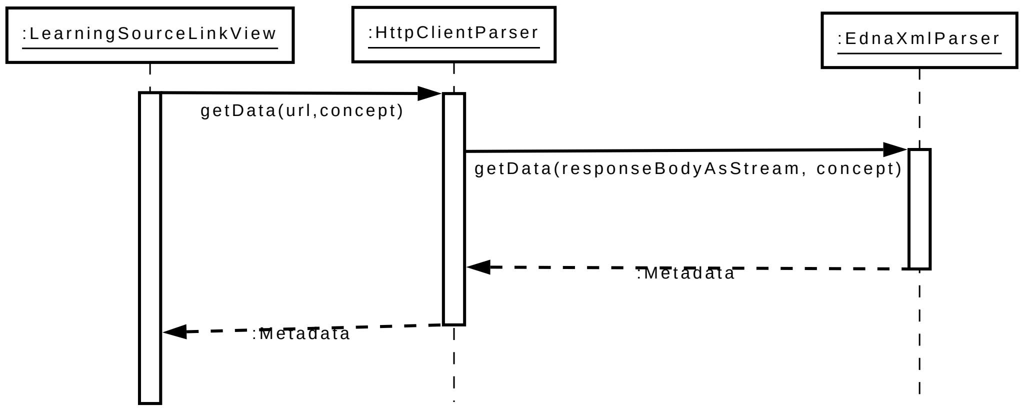


Figure 8.16: UML Sequence Diagram depicting how the EDNA LOR is queried using the screen-scraping approach

In figure 8.16, we have outlined the main classes involved in the screen-scraping approach used to query the EDNA LOR[5]. An instance of this class calls the *HttpClientParser*, which is responsible for screen-scraping LOR queries. The result of this screen-scrape is passed to the *EdnaXmlParser* to parse the result and build *Metadata* models, from the CAVIAr Learning Context Model, to be returned to the *LearningSourceLinkView*. The *LearningSourceLinkView* populates the metadata section of the LOR view, as outlined in section 8.7.3, allowing the course creator to analyse the details of the LO. To allow for more LORs to be integrated into MIKAEL the Abstract Factory pattern [Gamma et al., 1995] is used. This pattern enables delegation to a different parser depending on what LOR is scraped.

8.10.2 Service-based LOR Queries

There is currently a concerted effort to standardise how LORs are queried using a LOR interoperability specification such as the Simple Query Interface (SQI) [Simon et al., 2005]. Complementary to this approach is the development of a LOR query language, known as the ProLearn Query Language (PLQL) which has been developed specifically for the query needs of LORs [Ternier et al., 2008]. Here, we look at how these approaches can be used to query LORs using MIKAEL.

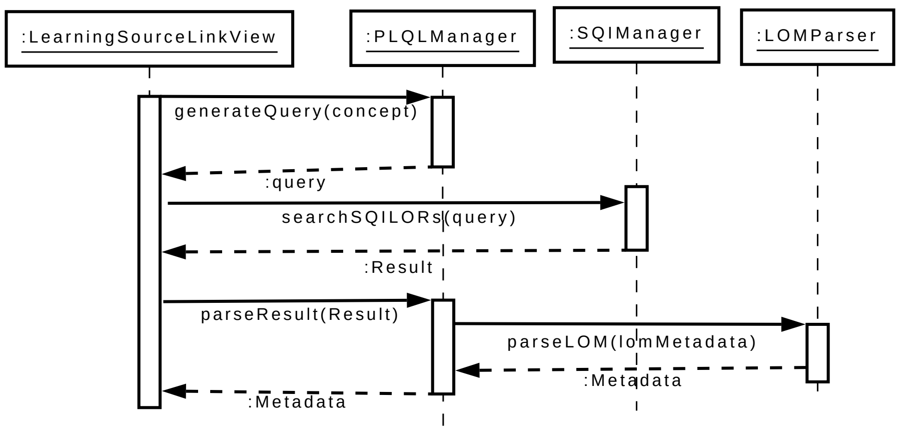


Figure 8.17: UML Sequence Diagram depicting how SQI-compliant LORs can be queried using PLQL

[5]`http://www.edna.edu.au/`

Figure 8.17 outlines an UML Sequence Diagram outlining the classes involved in a SQI-based query of a LOR using MIKAEL. The Sequence Diagram describes a process where the *PLQLManager* generates a query. As PLQL has various levels of expressive power, the *PLQLManager* is responsible for generating PLQL with the correct expressivity level for the LOR(s) being used by the course creator. The query then uses the SQI manager, to wrap the PLQL in SQI and send it to the LOR(s) of choice. The result of this query is LOM definitions for LOs and is parsed by the *LOMParser*, generating CAVIAr *metadata* models.

We note that various standards have been used together here. The course creator may wish to use a different combination of standards and specifications, for example the LOR may return Dublin Core [DCMI, 2006] rather than LOM. To allow for this the Abstract Factory pattern [Gamma et al., 1995] has been employed here also, allowing for the parsing the different specifications and standards in MIKAEL to be delegated to specialised classes. This increases MIKAEL's maintainability as the parsing concern is separated into specialised classes.

8.11 Chapter Conclusion

In this chapter, we have described the MIKAEL tool. MIKAEL provides tool support for the CAVIAr-based courseware construction process, also outlined in this chapter. This courseware construction process allows for the definition of CAVIAr models for the purpose of courseware construction. As part of the courseware construction process the course creator can validate constructed courseware using the CAVIAr models.

MIKAEL provides a proof of concept tool for validating courseware using CAVIAr. The courseware construction process illustrates how the validation process activities in chapter 7 can be integrated into courseware construction, thereby relieving much of the burden of creating CAVIAr models solely for validation.

We note the courseware construction process can be aligned with the OMG's Model Driven Architecture (MDA) approach to MDE [Frankel, 2003]. Courseware is created using three levels of abstractions that progressively get more implementation oriented. Requirements defined in the Learning Context Model can be aligned to the Computation Independent Model (CIM), the Courseware Model is aligned to the Platform Independent Model (PIM), as the CAVIAr Courseware Model is standard/specification neutral. The Courseware Model is then used to create courseware in terms of a courseware specification, a Platform Specific Model (PSM), where the courseware specification is the “platform”.

MIKAEL was developed using the Eclipse framework, using EMF to manage the CAVIAr models. User Interfaces were developed using GMF to help the author to define the Domain and Courseware Models. Eclipse perspectives were designed around the major steps in the courseware construction process, defined in figure 8.1. Model transformation technology was used to define how courseware requirements in the Learning Context Model could be used to generate a Courseware Model. We also demonstrated how CAVIAr interoperability is achieved through model transformation technology. The validation of courseware is achieved by integrating an OCL Checker with the EMF validation framework. This allows the course creator to validate courseware in MIKAEL using OCL. MIKAEL can be easily integrated with LORs, using the LOR manager component.

The MIKAEL Model Transformation Manager component uses ATL to define model transformations. One issue we had with ATL was that it is not possible to copy source model constructs to the target model if those source model constructs were used as the basis for a transformation. This was a problem when generating a Courseware Model as when generating a Courseware Model the Learning Context Model must also be available for validation. In generating the Courseware Model, the Learning Context Model constructs were used in the transformation and therefore could not be copied to the newly generated Courseware Model. To overcome this problem a second transformation was defined to merge the Learning Context Model with the newly created Courseware Model. This ATL transformation definition can also be found in appendix B.

In chapter 9 we will evaluate the use of MIKAEL for courseware construction in terms of its user acceptance. MIKAEL was used extensively in user trials conducted as part of the CAVIAr evaluation.

Chapter 9

Evaluation

9.1 Introduction

The evaluation of our research must be viewed in the context of our original research problem and how we have proposed to address this problem. In chapter 1 we identified the need for automated courseware validation as part of the courseware construction process, where validation ensures that constructed courseware satisfies pedagogical and non-pedagogical requirements defined by the course creator.

We identified the following **research challenges** in order to achieve validation of courseware in section 1.2:

1. Identify the data available for courseware validation pre-delivery in terms of courseware requirements as defined by those involved in courseware construction.

2. Investigate how the courseware requirements can be represented explicitly.

3. Develop an approach to validate courseware using the courseware requirements. This approach should be optimised towards personalised and personalisable courseware, as it is a major trend in Technology Enhanced Learning (TEL) [Wade and Ashman, 2007].

4. Investigate how courseware validation can be integrated with existing courseware construction tools.

5. Design and implement a proof of concept application that clearly validates our research in terms of its feasibility.

6. Evaluate the research by investigating user acceptance of courseware validation within courseware construction in general and our approach to validation in particular. User acceptance looks at the following:

   - Usability - The validation approach and its tool support must be usable by the course creator.
   - Cost Effectiveness - The validation approach must be cost effective in terms of course creator effort.
   - Effectiveness - The approach captures courseware problems and requirements effectively. The validation approach must be able to validate problems in courseware that the course creator deems to be important.
   - Modifiability - The course creator must be able to customise the validation constraints criteria according to his or her own requirements. Validation support must be flexible enough to be integrated with the TEL specifications used by the course creator in courseware construction.
   - Performance - The performance of our validation approach must be acceptable when compared with the state of the art.

In chapter 4 the courseware construction concerns were identified. The courseware construction concerns represent the data available for courseware validation pre-delivery. This addresses research challenge 1. Based on the courseware construction concerns we defined the Courseware Authoring Validation Information Architecture (CAVIAr), in chapter 5 and 6, as a set of models and modelling constraints that allow for the explicit representation of courseware construction concerns and courseware requirements, addressing research challenge 2. The activities involved in validating courseware using CAVIAr were defined in chapter 7, satisfying research challenge 3. The design and implementation of a proof of concept software application was described in chapter 8. This software, known as MIKAEL, allowed for courseware construction and validation based on CAVIAr. MIKAEL demonstrated the feasibility of CAVIAr courseware validation and satisfies research challenge 5. In chapter 8 we also outlined how CAVIAr-based validation could be integrated with the state of the art through model transformation technology. This addresses research challenge 4.

In this chapter we address our only outstanding research challenge, to evaluate the user acceptance of our approach to courseware validation (challenge 6). User acceptance is assessed by evaluating the effectiveness of our approach to capture courseware requirements, the cost effectiveness of the approach, its usability, modifiability and performance, as defined in the research challenges.

In the next section (section 9.2), we will outline the evaluation strategies that we have used to evaluate user acceptance. After describing the evaluation strategies we will evaluate each user acceptance characteristic using one or more of the evaluation strategies. The user acceptance characteristics were addressed as follows:

- Effectiveness - Section 9.5.
- Cost Effectiveness - Section 9.4.
- Usability - Section 9.3.
- Modifiability - Section 9.6.
- Performance - Section 9.7.

We conclude the chapter by summarising our findings in section 9.8.

9.2 Evaluation Strategies

There were three principle strategies taken to evaluate the research documented in this thesis. These were:

- Empirical Study - An empirical study was carried out through the MIKAEL user trials. The MIKAEL user trials looked at usability, course creator perceived cost and effectiveness of CAVIAr validation and were then asked questions through questionnaires. Assessing user satisfaction did not include formal user testing.

- Analytical Study - We methodically analysed the suitability and effectiveness of each CAVIAr data model defined in this thesis.
- Comparative Study - Modifiability and performance were evaluated through a comparative study with the state of the art.

9.2.1 Empirical Study

As part of the evaluation of the MIKAEL tool, described in section 8, we carried out user trials with course creators from industry and academia. The aim of these user trials were as follows:

- Evaluate the usability of the MIKAEL tool in terms of its user interface.
- Evaluate the usability of the CAVIAr models.
- Evaluate CAVIAr-based validation effectiveness as perceived by the course creator.
- Evaluate CAVIAr with respect to its effectiveness in capturing the courseware construction concerns to allow for courseware construction.
- Evaluate CAVIAr with respect to its perceived efficiency when used in courseware construction (such as that found in MIKAEL).

We will describe the MIKAEL user trials in detail in section 9.3. The user trial results are then described in section 9.3, 9.5 and 9.4 as part of our evaluation of CAVIAr usability, effectiveness and cost effectiveness respectively.

9.2.2 Analytical Study

In evaluating the effectiveness of the CAVIAr data models we analysed the CAVIAr models in terms of two types of criteria, structural criteria and semantic criteria. Structural criteria looks to evaluate the *soundness* and *completeness* of each CAVIAr metamodel. Semantic criteria takes into consideration application-specific information in order to evaluate a given CAVIAr data model. In section 9.5.1 we provide details of the approach used to analytically evaluate the CAVIAr data models, and provide results from our evaluation.

9.2.3 Comparison with the State of the Art

We have evaluated CAVIAr validation in terms of its modifiability (section 9.6) and performance (section 9.7) by comparing it with the state of the art.

We evaluated modifiability in CAVIAr validation using a scenario-based approach known as ALMA (Architecture Level Modifiability Analysis) [Bengtsson et al., 2004]. ALMA is used to compare the modifiability of CAVIAr-based information architecture with the state of the art in courseware interoperability and courseware validation. We consider the modification of the following key validation concerns:

- A change to the native domain knowledge specification that is used for the CAVIAr Domain Model definition.
- The courseware specification used is changed.
- The validation criteria must be changed.

In evaluating CAVIAr validation performance, we compare the simulation-based validation approach, the primary approach used in the state of the art, with CAVIAr's constraint-based approach in terms of its time complexity.

9.3 Usability

In this section we give details on the MIKAEL user trials that we have conducted and present the results of the MIKAEL user trial related to usability. The following subsection, section 9.3.1, outlines how the user trials were conducted. In subsection 9.3.2 we detail the participant demographic that took part in the user trials. In section 9.3.3 we present the usability results from the user trials. Other results from the MIKAEL user trials will be presented in subsection 9.4 (cost effectiveness) and section 9.5 (CAVIAr effectiveness).

9.3.1 MIKAEL User Trials Overview

The MIKAEL user trials were conducted as a series of one-to-one tutorials on the MIKAEL tool with course creators (details of the participants will be covered in the next subsection). In these tutorials the participant was shown how to create new courseware in accordance with the courseware requirements for the case-study in chapter 7, using the MIKAEL tool. We did not allow the course creator to manipulate the courseware construction directly, as it was not feasible to teach the functionality of MIKAEL as well as demonstrate courseware construction within the time constraints of the tutorial. The format of the tutorials involved intermittently asking the participant questions while constructing courseware using MIKAEL. We followed this format so that the questions were about the courseware construction activities that had just been witnessed. This limited the cognitive demand of the survey allowing the course creator to concentrate on how MIKAEL is used for courseware construction. The survey questionnaire we used in our experiment can be found in appendix C. The evaluation survey was predominantly quantitative in nature but had some qualitative questions. The user trial structure is detailed below:

1. Background - We initially presented an introduction of the courseware construction process, as defined in chapter 8.2, to participants. In this presentation we outlined how MIKAEL supports the courseware construction process by providing a user interface (Eclipse perspective, defined in section 8.7) for each of the main courseware construction activities.

2. Survey Session - The participant answered questions related to the courseware construction process to gauge their understanding - results in section 9.4.

3. Domain Model Editing - This part of the tutorial consisted of importing a SKOS ontology into MIKAEL and editing it according to the databases courseware requirements in section 7.2. The participant was also shown how to search for LOs using MIKAEL and how to add a LO to a concept. At this stage we also showed to the participant how to add conceptual sequencing constraints between concepts in the Domain Model.

4. Survey Session - The participant answered questions related to the Domain Model - results in section 9.3.3.1.

5. Learning Context Model Editing - This demonstrated how a learner stereotype is defined in terms of the courseware learning goals and presumed knowledge.

6. Survey Session - The participant answered questions related to the Learning Context Model - results in section 9.3.3.2.

7. Courseware Model Editing - Participants were shown how to generate a Courseware Model from the requirements defined in the Learning Context Model. We then explained to the participant how the Courseware Model is generated through a mapping from the Learning Context Model to the Courseware Model and that the courseware generation definition is independent of MIKAEL's programming code. We demonstrated how to edit the generated Courseware Model, and how requirements defined in the Learning Context Model are mapped to the Courseware Model. We explained to the participant how personalisation is defined using entry learner conditions. Participants were also shown how to add/edit/delete LOs to/from courseware topics.

8. Survey Session - The Participant answered questions related to Courseware Model editing - results in section 9.3.3.3.

9. Courseware Validation - Participants were shown how courseware validation is achieved by firstly showing them an editable CAVIAr Validation Model defined in OCL and then validating courseware. We showed participants that the Validation Model is editable to ensure that they understood that the Validation Model is not part of the MIKAEL's programming logic but is a separate and editable definition of what is correct for a given courseware definition. The CAVIAr Validation Model used a constraint from each of the type of validation rule defined in section 6. The courseware created in the previous steps was then validated and the validation results were shown to the participant for inspection.

10. Survey Session - The participant answered questions related to the effectiveness (results in section 9.5) and usability (usability results in section 9.3.3.4) of validation.

11. Concluding questions - After the courseware was created and validated, participants were shown how to export the courseware created into a SCORM package. This SCORM package was then imported into the Moodle LMS. This allowed the course creator to see how courseware developed using MIKAEL is delivered.

12. Survey session - The participant answered general questions regarding the usability (results in section 9.3.3.4) and effectiveness (results in section 9.5) of CAVIAr-based courseware construction. We also asked the course creator about the potential time and cost saving (if any) that could be achieved by using MIKAEL for courseware construction (results in section 9.4).

All questions were asked orally, allowing the participant to focus visually on the trial task. Each "survey session" consisted of a set of closed questions, where the participant answered using a five point Likert scale [Likert, 1932] and then one open-ended question that allowed the participant specify additional comments about the previous courseware construction stage. At the end of the trial the participant could add comments about the MIKAEL tool or CAVIAr-based courseware construction, in general. It was not compulsory to answer any question.

9.3.2 MIKAEL User Trials Participant Details

Courseware construction is an expert activity, these experts are collectively known as course creators, as defined in section 4.2. In our user trials we assembled a sample set of course creators (n=14). Participants in the sample had a wide variety of backgrounds:

- Academic.
  - School of Computing in Dublin City University.
  - School of Computing in the National College of Ireland.
- Industry.

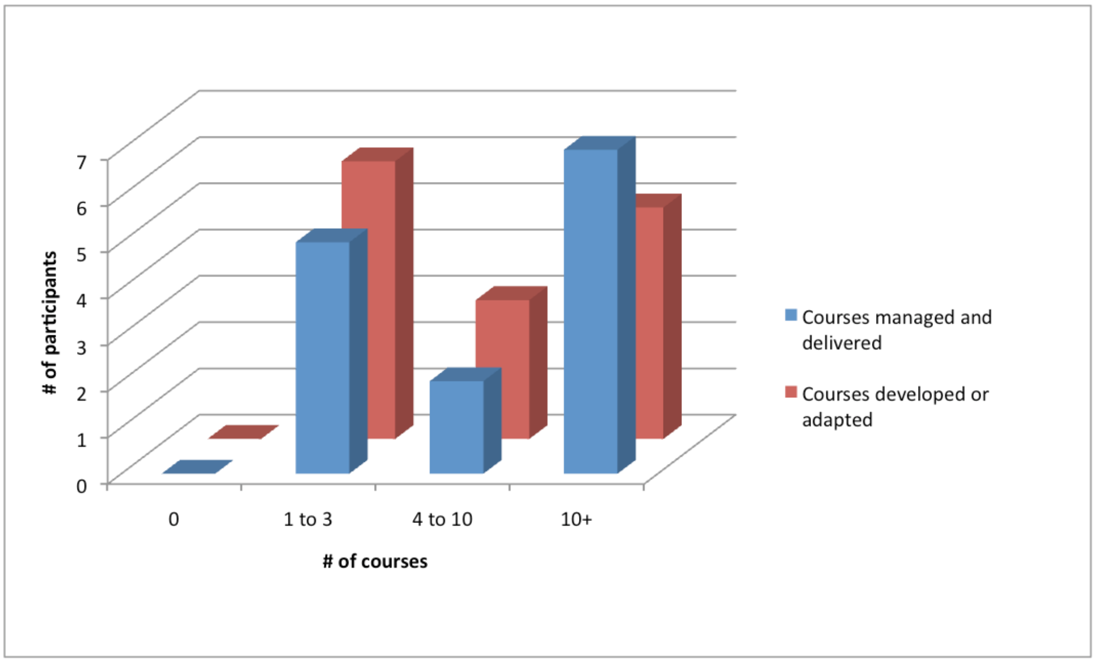


Figure 9.1: Breakdown of participant knowledge

- The financial services sector.
- The IT sector.
- The telecoms sector.

In figure 9.1, we have illustrated the knowledge breakdown of the trial participants as defined by the participants themselves. All considered themselves to be familiar or an expert in e-learning, while 10 participants (71%) considered themselves familiar or expert in e-learning authoring, 12 participants (85%) stated they were familiar with personalised e-learning, but none considered themselves an expert in personalised e-learning.

Figure 9.2 outlines the participants experience in delivering courses and also in creating or adapting courses. This shows that all of the participants in our trial have delivered or adapted courses and that the majority of the participants are experts, doing it more than ten times.

Through the survey we also assessed the types of courses the participants had created or adapted and found that seven participants had created/adapted courses for academia, ten had

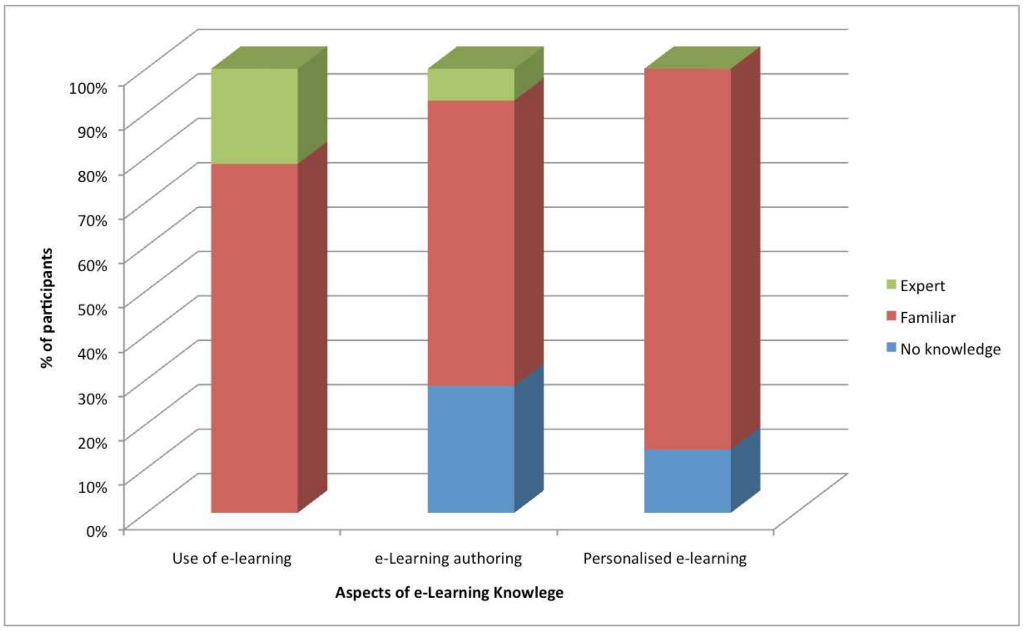


Figure 9.2: The number of courses participants have managed and delivered, and the number of courses participants have adapted or created

created/adapted courses for industry training, while three participants had created/adapted professional training courses, such as a project management course.

9.3.3 MIKAEL User Trial Survey Results

In this section we will outline the MIKAEL user trial survey results related to usability.

9.3.3.1 Domain Model Editing

After demonstrating importing and editing a Domain Model in MIKAEL we asked the participants a number of questions. Questions can be found in Appendix C, section 3. In figure 9.3 we outline the results from this survey related to usability.

9.3.3.2 Learning Context Model Editing

After the Participants are shown how to define a Learning Context Model. The participant was then asked questions on defining a Learning Context Model. Questions can be found in Appendix C, part 4. Figure 9.4 outlines the result from this survey related to usability.

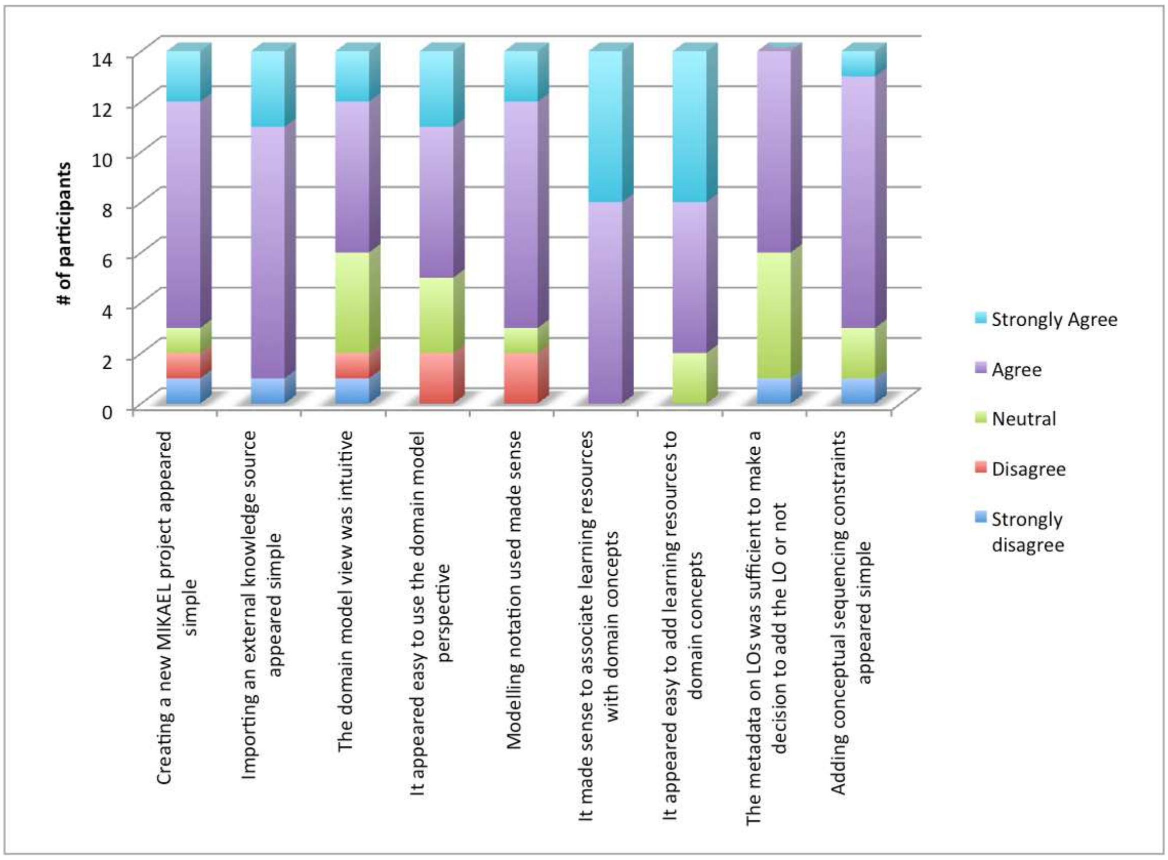


Figure 9.3: Domain Model editing survey results

9.3.3.3 Courseware Model Editing

After demonstrating to the participants how a Courseware Model is generated from the Learning Context Model we demonstrated MIKAEL's courseware model editing functionality. The participants were then asked about their sentiment towards the Courseware Model and how it is defined. Survey questions can be found in Appendix C, part 5. In figure 9.5 we outline the results of this survey related to usability.

9.3.3.4 Post-validation

After validating courseware, we asked the participant a variety of question on how they felt about CAVIAr courseware validation. These questions can be found in Appendix C, part 6 and 7. The survey results related to usability are outlined in figure 9.6.

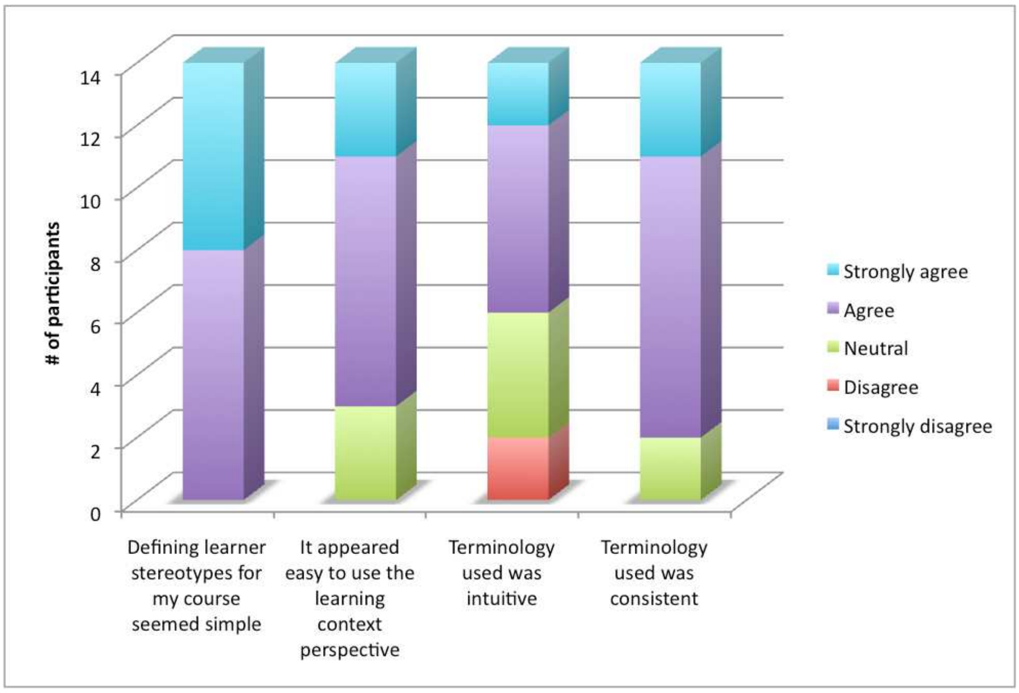


Figure 9.4: Learning Context Model editing survey results

9.3.4 Discussion

As outlined in the graph in figure 9.3, participants were positive about Domain Model editing. Most participants thought creating a MIKAEL project and importing external knowledge was simple. Domain model editing usability is not a major concern as the majority of participants found the domain model view intuitive and the perspective appeared easy to use. Using the Domain Model to drive a search for LOs was something that made sense to all participants, with 85% of participants indicating they thought it was easy to add a LO to a concept. The majority of participants found the metadata displayed about a given LO gave enough information to decide whether or not to add a LO to a concept. Three participants (23%) mentioned that it would be useful for MIKAEL to provide a way for the course creator to preview a learning resource. One participant, in particular, noted “It would be useful to allow users to fully review a learning resource before adding it by allowing them to link directly to the resource from within the tool”.

All participants thought creating a learner stereotype in MIKAEL was simple (section

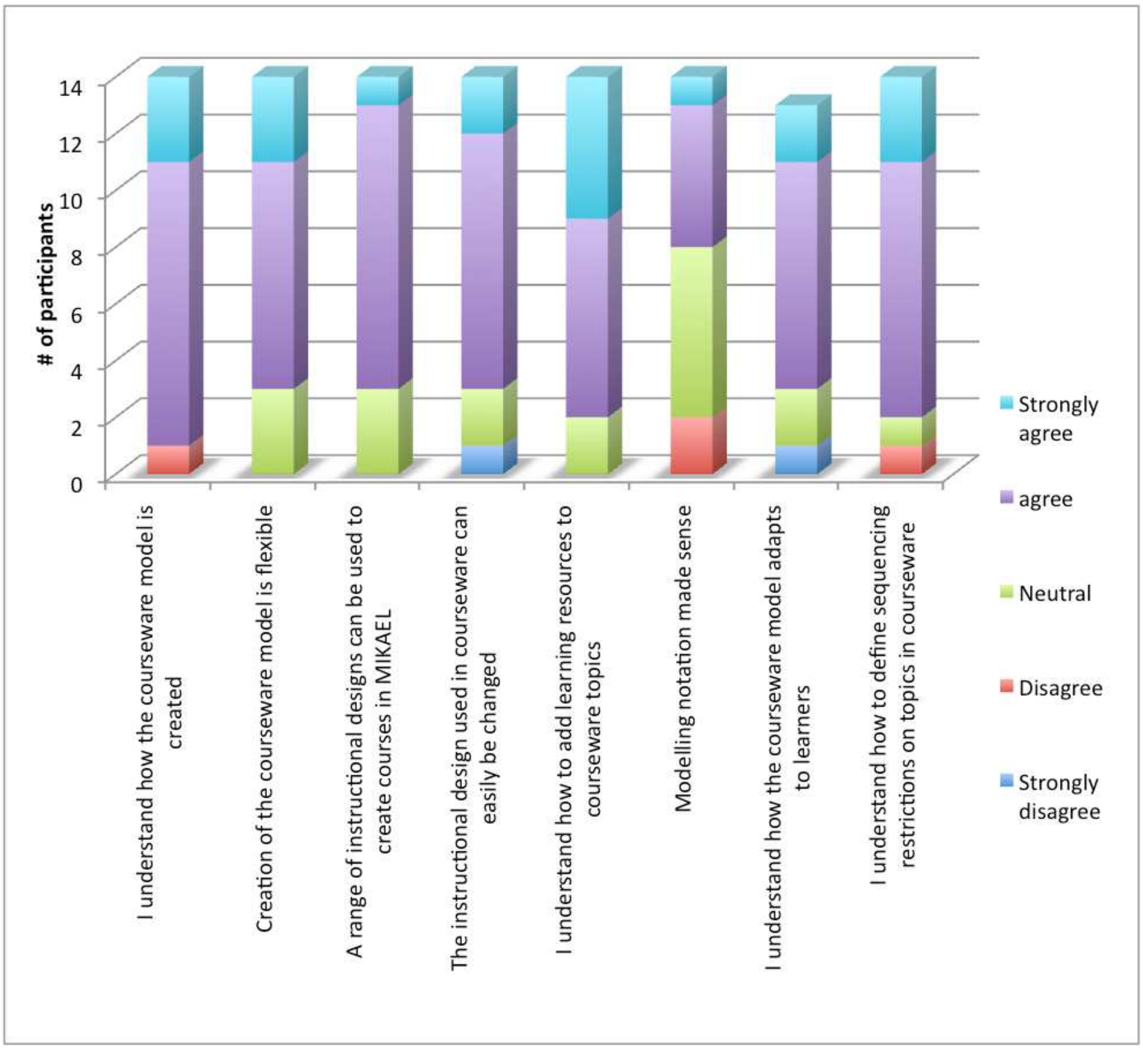


Figure 9.5: Courseware model editing survey results

9.3.3.2), with one stating that it was a good way to represent the dynamics of today's courseware requirements. Nearly all found the learning context perspective intuitive. Terminology used was found to be intuitive and consistent, by the majority of the participants. An example of this kind of terminology consistency is being able to relate concepts in the Learning Context Model with concepts in the Domain Model. We believe that these results to be a vote of confidence in the underlying CAVIAr Learning Context Model as a tool to define courseware requirements.

We surveyed the participants on their opinion on how MIKAEL allows for the course

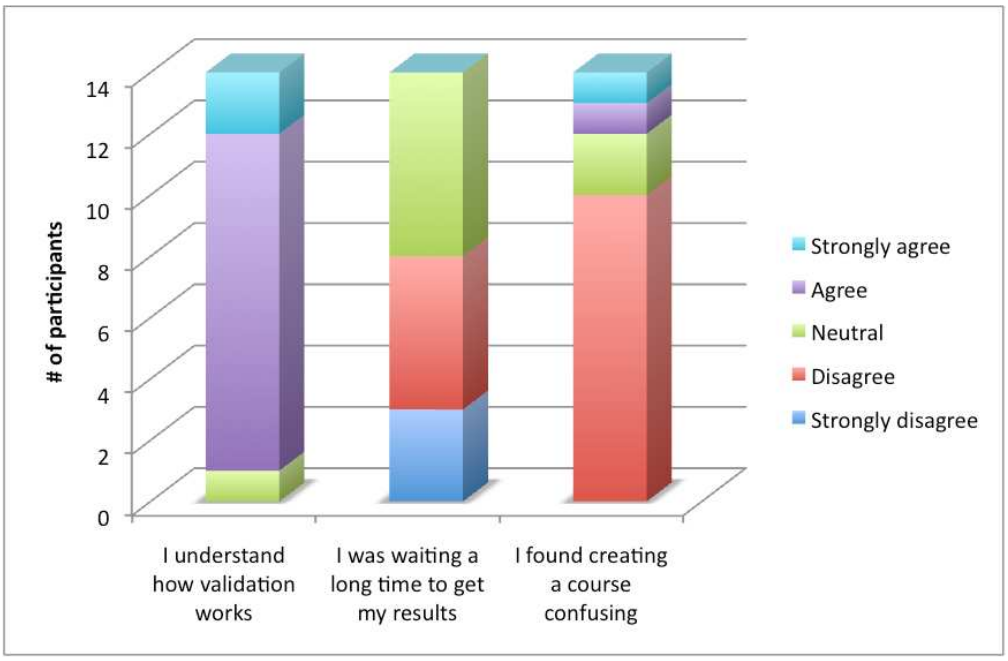


Figure 9.6: Usability survey results after validating courseware

creator to edit a given courseware definition by editing the CAVIAr Courseware Model, the results of this can be found in section 9.3.3.3. We found that most participants understood how the Courseware Model was generated at a high level (93%). Most participants also thought MIKAEL's approach to creating a Courseware Model was flexible and recognised that the instructional design being used in the transformation definition could be changed. Participants felt they understood how to edit the learning resources used in a given topic. Two participants gave a negative result regarding the intuitiveness of the modelling notation used for the Courseware Model, with the remainder giving a neutral (6 participants) or positive answer (6 participants). The majority of participants gave positive answers regarding their understanding on how courseware adapts to a learner and how to define sequencing restrictions on the Courseware Model, although one person gave a negative answer for each. We believe that this was due to a misunderstanding with regard to the CAVIAr modelling notation used in MIKAEL. One participant noted "the stereotype name on *entrylearners* would be useful", and another said the "arrow [is] pointing the wrong way for the seq_after [topic] relationship" indicating that there was some confusion with defining Courseware

Model constructs.

Nearly all participants (except for one who gave a neutral response) understood how validation worked. No participant felt that validation took an unreasonable amount of time to complete (seven to ten seconds). Although this result would be influenced on the complexity of constraints defined and the size of the CAVIAr models. This is probably because validation is an irregular (in many cases once-off) courseware construction activity.

From these results we can conclude that the course creator found editing the CAVIAr models intuitive during courseware construction in terms of:

- The CAVIAr courseware construction process.
- Using the CAVIAr models for courseware construction.
- The implementation of the CAVIAr models in MIKAEL.
- The validation of constructed courseware using CAVIAr through MIKAEL.

9.4 Cost Effectiveness

In this section we evaluate if there are cost implications associated with using a CAVIAr-based courseware construction process. To do this we surveyed course creator's sentiment towards using MIKAEL for courseware creation with regard to cost implications. Cost was judged in terms of the time and capital investment required for courseware creation. The survey questions looked at the following:

- Is there a significant learning curve associated with initially using CAVIAr-based courseware construction?
- Did the course creator find MIKAEL useful?
- Does the course creator think MIKAEL is time and cost effective?
- Does the course creator think that it will be easier for him or her to reuse existing learning material with MIKAEL?

9.4.1 MIKAEL User Trial Survey Results

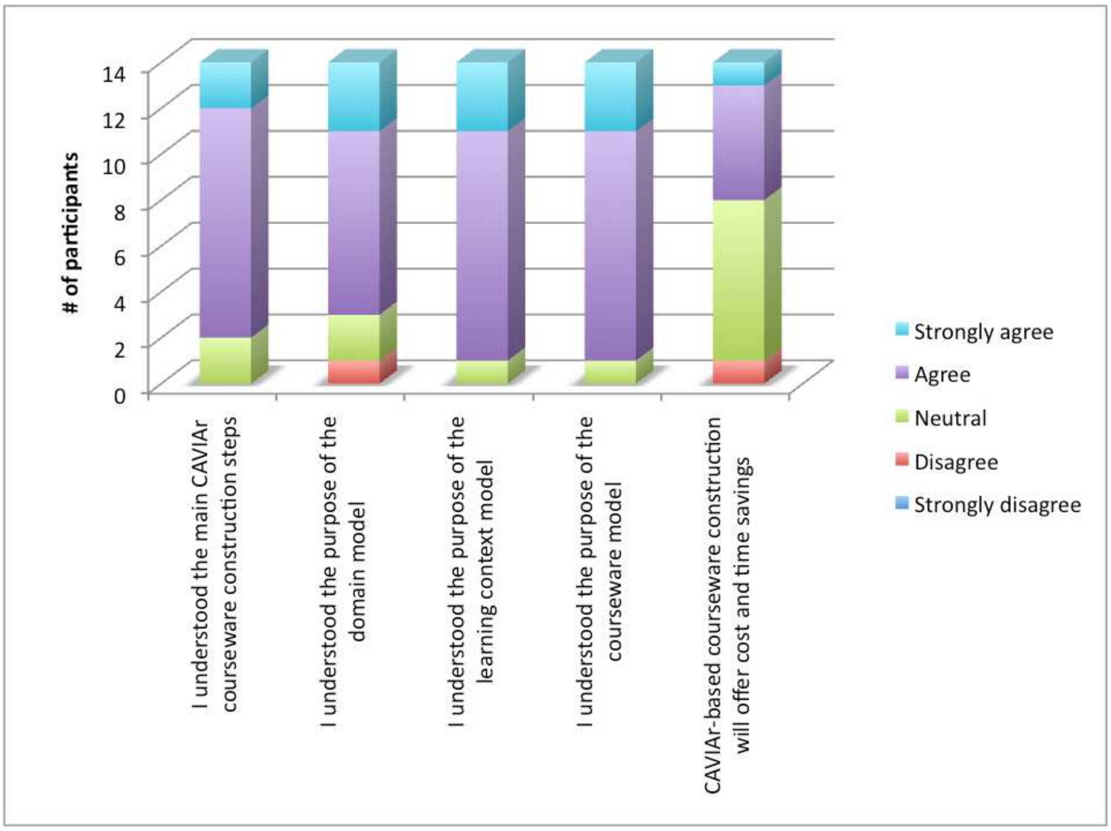


Figure 9.7: Participant understanding of the main aspects to CAVIAr and CAVIAr-based courseware construction through MIKAEL

After an initial presentation on how MIKAEL constructed courseware by building up CAVIAr models we asked the participants questions to ensure they understood how CAVIAr-based courseware construction worked and to investigate if the participant could see potential for cost savings in using MIKAEL. The results of this are presented in figure 9.7.

Figure 9.8 outlines the results of the MIKAEL survey associated with the cost and time of using MIKAEL as perceived by the course creator. These survey questions were asked at the end of the MIKAEL user trial.

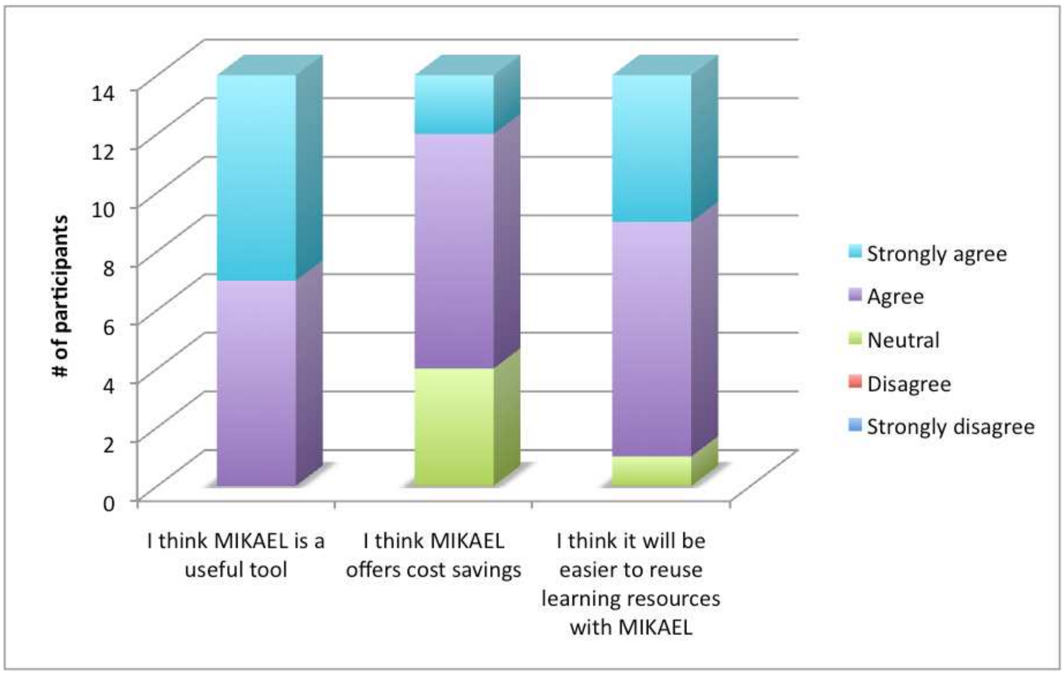


Figure 9.8: MIKAEL survey results relating to the cost effectiveness of MIKAEL

9.4.2 Discussion

The questions after doing the MIKAEL user trial show that all participants found MIKAEL useful. Most participants (71%) thought MIKAEL would produce courseware construction cost savings, with the remainder of the participants giving a neutral answer. This is a marked improvement on the 43% who thought MIKAEL would produce cost savings prior to seeing the MIKAEL demo (figure 9.7). Participants see MIKAEL as an enabler for the reuse of existing learning resources. From this we can conclude that the course creator does see cost benefits associated with reuse in MIKAEL. Figure 9.8 also indicates that course creators see MIKAEL as a useful tool and they think it is easy to reuse learning resources with MIKAEL.

We also believe MIKAEL could offer significant cost savings indirectly due to its relatively low training requirements. This is based on the fact that after just a brief introduction on how the tool works the majority of course creators indicated that they understood each element of CAVIAr-based courseware construction, as indicated in the graph in figure 9.7.

9.5 Effectiveness

We evaluate the effectiveness of CAVIAr in terms of an:

- Analytical evaluation of CAVIAr for defining the course construction concerns and represent constructed courseware.

- Empirical evaluation of course creator's perceived effectiveness of the CAVIAr metamodels for courseware validation, evaluated through the MIKAEL user trials outlined in section 9.3.

9.5.1 Analytical Evaluation of the CAVIAr Data Models

In evaluating the effectiveness of the CAVIAr data models we define two types of criteria, structural criteria and semantic criteria. Structural criteria looks to evaluate the *soundness* and *completeness* of the CAVIAr metamodels. Semantic criteria takes into consideration application-specific information in order to evaluate a data model. Although inconsistencies with certain structural criteria definitions may be disregarded, the CAVIAr metamodel must satisfy its semantic criteria.

Guizzardi et al. outline how to determine if a Domain-Specific Modelling Language (DSML) is sound and complete by comparing the metamodel with a well-established domain conceptualisation, such as an ontology [Guizzardi et al., 2005]. They define a modelling language as *sound* when every modelling primitive can be represented in the domain conceptualisation and they define a language as *complete* when every concept in the domain conceptualisation is represented in the modelling language. To establish the soundness and completeness of the CAVIAr metamodels, we mapped each metamodel to a well-defined conceptualisation that represents that metamodel's course construction concern. We define a metamodel as *sound* when all elements, or groups of elements, in the metamodel can be mapped to an element, or group of elements, in the domain conceptualisation. We define a metamodel as *complete* when all elements, or groups of elements, in the domain conceptualisation can be mapped to an element, or group of elements, in the metamodel.

To establish if an *incomplete* metamodel is *sufficient* to satisfy the information needs of validation we define a minimal level of completeness that the metamodel must achieve. Should a metamodel be found to be *unsound*, we establish if there is adequate rationale for the inclusion of extra modelling constructs in the metamodel not in the domain conceptualisation using the literature.

9.5.1.1 Domain Model

The CAVIAr Domain Model can be mapped to the Simple Knowledge Organisation Structure (SKOS). We use SKOS as it is a "light weight, intuitive language for developing and sharing new knowledge organization systems" [Miles and Bechhofer, 2009]. SKOS aims to provide a simple knowledge structure similar to that which was needed for the CAVIAr Domain Model, which as stated is "a knowledge structure, describing concepts and relationships between concepts" (section 5.3). SKOS is an established conceptualisation for representing conceptual structures that is currently undergoing World Wide Web Consortium (W3C) standardisation.

Section 5.3.3 illustrated how each of the CAVIAr Domain Model constructs can be mapped to a SKOS modelling construct, meaning the CAVIAr Domain Model's metamodel is sound. Although it is sound, the Domain Model is incomplete as the following SKOS modelling constructs have no representation in the CAVIAr Domain Model:

- *skos:hiddenLabel* - A concept label hidden from the user.
- *skos:scopeNote* - Intended meaning of a concept.
- *skos:definition* - Provides complete explanation for a concept.
- *skos:example* - Supplies an example for a concept.
- *skos:historyNote* - Outlines changes to the concept meaning or form.
- *skos:editorialNote* - Ontology of administration notes.
- *skos:changeNote* - Fine-grained changes to concepts.

- *skos:Collection* - Collection of concepts, where membership is defined by *skos:member*.
- *skos:orderedCollection* - Ordered collection of concepts.
- *skos:broaderTransitive* - Transitive relationship for skos:broader.
- *skos:narrowerTransitive* - Transitive relationship for skos:narrower.

In section 5.3, we have stated that the Domain Model need only be a simple knowledge structure describing concepts and the relationships between concepts. Therefore the CAVIAr Domain Model's metamodel is *sufficient* for courseware validation if it can be mapped to the following SKOS modelling constructs that allow for simple conceptual structures: *skos:concept*, *skos:prefLabel*, *skos:altLabel* and *skos:semanticRelationship*.

9.5.1.2 Learning Context Model

The Learning Context Model represents the requirements of courseware. It does this by constraining the Domain Model in two ways, using conceptual instructional constraints and by defining learner stereotypes. A well-established conceptualisation designed to constrain a Domain Model for the purposes of defining e-learning requirements is the "goal and constraints model" in LAOS. LAOS is covered in detail in section 3.2.2.1.

LAOS is based on the AHAM architecture [DeBra et al., 1999]. AHAM allows for concept links of type *prerequisite* to define conceptual sequencing constraints. A one-to-one mapping can be made between the AHAM concept link (relationship) in AHAM to the prerequisite relationship in the CAVIAr Learning Context Model definition.

In LAOS domain model concepts can be assigned weights by the course creator. This mirrors the assignment of a *knowledgeLevel* to a *knowledgeElement* in the CAVIAr Learning Context Model. The CAVIAr *knowledgeElement* also allows the course creator to define a *knowledgeType*. The course creator cannot assign a *knowledgeType* in LAOS. In LAOS a goal is not explicitly defined as it is in CAVIAr, but is implicitly defined as all domain model concepts are goal concepts through the goal "*AND*" link between domain model concepts. If there are alternative goal concepts in the domain model the *AND* link can be changed to an *OR* link. Both these LAOS goal and constraint concept links map directly to the CAVIAr goal constructs in the Learning Context Model.

The LAOS goal and constraints model does not allow for the definition of separate learner stereotypes, but this can be defined in the LAOS user model definition. LAOS user models are defined in terms of a goal and constraint model. Learner stereotypes in the CAVIAr Learning Context Model can be mapped to the LAOS user model. *Presumed-Knowledge* can also be expressed for each learner in the LAOS user model, by defining an initial knowledge level for an individual or a group of learners.

We can therefore conclude that all the elements in LAOS can be expressed with the CAVIAr Learning Context Model. This means the Learning Context Model is complete. However, all elements in the CAVIAr Learning Context Model cannot be represented in the LAOS goal and constraints model meaning that the CAVIAr Learning Context Model is unsound. This is due to the addition of a *knowledgeType* element when defining knowledge in the CAVIAr Learning Context Model. We argue that a *knowledgeType* is necessary in order to represent the minimal attributes of knowledge such as a learning outcome, as defined by Gagné et al. in [Gagné et al., 2005].

9.5.1.3 Learning Resource Model

As we have outlined in section 5.5.3, the IEEE LOM standard [IEEE LTSC, 2002] is used as a domain conceptualisation for the representation of LOs in the CAVIAr Learning Resource Model. We have used LOM as it is an IEEE standard for describing learning resources. The *LO* definition in the CAVIAr Learning Resource Model is based on the IEEE LOM standard. There is a one-to-one mapping of IEEE LOM elements to CAVIAr *metadata* constructs in the Learning Resource Model.

In section 5.5.3, we also outlined a one-to-one mapping between the *LO* and *Service* elements in the CAVIAr Learning Resource Model and the LO and service definition in the IMS LD specification. We conclude that the CAVIAr Learning Resource Model representation of LOs is both sound and complete.

9.5.1.4 Courseware Model

The Courseware Model can be mapped to either of the main TEL courseware specifications ADL SCORM 2004 [ADL, 2004] or IMS LD [IMS, 2003b]. To establish the soundness and completeness of the CAVIAr Courseware Model we compare it with the IMS LD specification. We have chosen to use the IMS LD specification as it is the more flexible and progressive specification, for example, allowing for the definition of different actors in the learning process and allowing for the modelling of parallel learning activities.

In section 5.6.3, we briefly looked, at a high level, how the CAVIAr Courseware Model can be mapped to the IMS LD specification. The section described how each of the CAVIAr Courseware Model constructs can be mapped to an IMS LD modelling construct. We conclude from this that the CAVIAr Courseware Model appears to be sound. Here, we look at the IMS LD modelling constructs to see if they can be mapped to a CAVIAr courseware modelling construct or group of constructs. This will establish the completeness of the CAVIAr Courseware Model.

The IMS LD specification uses a theatre script metaphor [Koper, 2005]. Each of the script components (play and act) can be defined as CAVIAr courseware topics at different aggregation levels. A *method* in IMS LD co-ordinates *plays* by defining them in terms of a *learning objective* and *prerequisite*. These constructs cannot be mapped to CAVIAr Courseware Model constructs but can be mapped to the learning *goal* and *presumed knowledge* constructs in the CAVIAr Learning Context Model. *Conditions* can be defined on a *method* in IMS LD, this can be mapped to CAVIAr *entry learner* conditions that are defined based on *knowledge elements*. One of the core concepts in IMS LD is that everybody in the learning process gets a role, either a *staff* nor *learner* role. In CAVIAr, *topics* can be defined as suitable for particular learner roles through and *entry learner* conditions, which are based on a *learner stereotype* but there is no way to define *staff* roles in the CAVIAr Courseware Model.

In IMS LD there are two types of activities, learning activities and support activities. The CAVIAr Courseware Model has no way to distinguish between these two activities.

Activities exist within an IMS LD *activity-structure*. A *role* performs an *activity* within an IMS LD *act*. Learning activities can be mapped to *resources* in the CAVIAr Courseware Model. There is no resource structure in CAVIAr to map the *activity-structure* construct in IMS LD to. An *outcome* can be defined for an *activity* in IMS LD. This can then be used to trigger a *notification*. It is not possible to define an explicit *outcome* for a *topic* in the CAVIAr Courseware Model or a *notification*. A learning environment is defined using *learning objects* and *services* in IMS LD. These can be mapped to a learning resource definition in the CAVIAr Courseware Model.

From this analysis we can conclude that the CAVIAr Courseware Model is incomplete as many of the IMS LD constructs cannot be represented in it.

The IMS LD specification is concerned with roles other than learner ones and also defining parallel learning events. The CAVIAr Courseware Model is, at this stage of its development, not concerned with these and other advanced features used in the IMS LD specification. Also there are elements in the IMS LD, such as *learning objective* and *pre-requisite*, which are featured in other CAVIAr models (Learning Context Model) and as such do not require representation in the CAVIAr Courseware Model. We define a minimal set of IMS LD constructs necessary to deem the Courseware Model as *sufficient* for the purposes of courseware validation. These are as follows: *play, act, condition, learner (role), activity, environment, learning object* and *service*. All these IMS LD model elements can be represented in the CAVIAr Courseware Model as outlined above. We can therefore conclude that the CAVIAr Courseware Model is *sufficient* for courseware validation.

9.5.2 Empirical Evaluation of Course Creator's Perception of Validation Effectiveness

To evaluate how effective the course creator perceives CAVIAr validation we asked participants what they thought about courseware validation. This was done after the participant had validated the Courseware Model created during the user trials. The MIKAEL user trials were described in detail in section 9.3.

9.5.2.1 MIKAEL User Trial Survey Results

In figure 9.9, we outline the results related to how the course creator felt after validating the courseware with regard to the effectiveness of courseware validation.

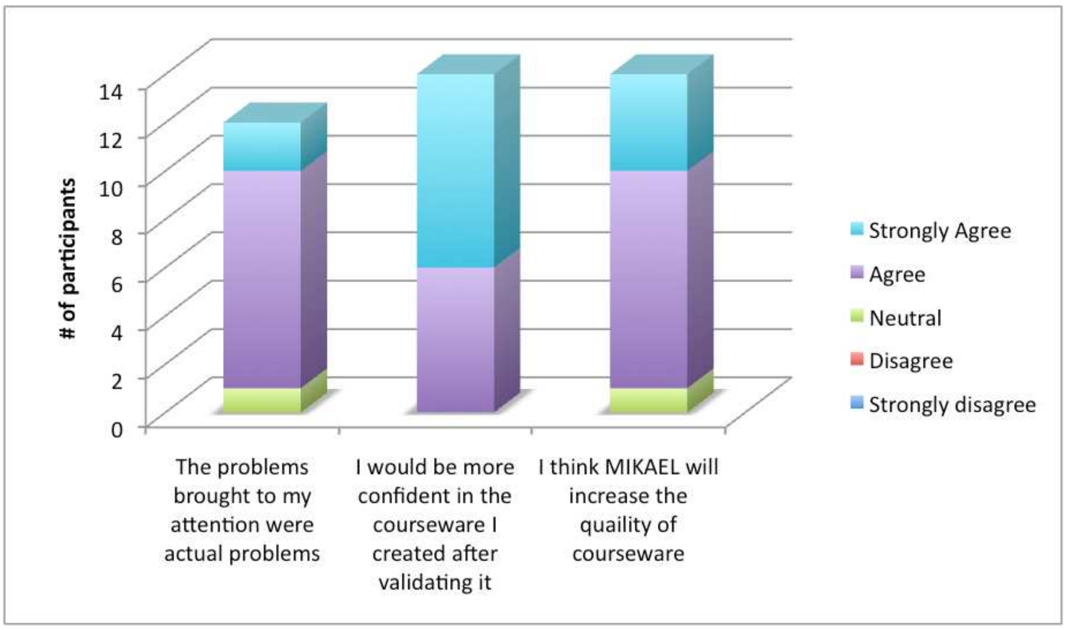


Figure 9.9: Course creator reflections on validation effectiveness

9.5.2.2 Discussion

The majority of participants thought that validation brings actual courseware problems to their attention. All participants agree that they would feel more confident about the courseware they had created after validating it, with 57% agreeing strongly. We also found that 93% of participants believe MIKAEL will improve the quality of courses (the remaining participants gave a neutral answer).

We found from the open-ended question after validation that many participants saw key advantages to CAVIAr validation, as validation constraints could be used to ensure that a given courseware meets some specified requirement, either for accreditation purposes or for a legislative requirement. Several Participants mentioned that course creators cannot be expected to define a Validation Model using OCL. In section 6.2.2 we outlined our initial efforts in defining a model-based DSML that allows the course creator to generate validation OCL using a more intuitive interface. We look at other approaches to tackling this issue in section 10.4.

9.6 Modifiability

Modifiability assesses how easily a CAVIAr implementation can be adapted to fit with a given course creator's requirements. To evaluate the modifiability of CAVIAr-based tools, we define a set of common maintenance scenarios that may be required in order to validate courseware. Using these scenarios we compare CAVIAr with the state of the art in terms of how easily they can be modified to cope with the given scenario. This approach is based on the ALMA (Architecture Level Modifiability Analysis) method [Bengtsson et al., 2004], a scenario-based method used to evaluate software architecture modifiability. ALMA is used to establish maintenance cost, assess risk and compare competing software architectures. In our research we use ALMA in a comparative setting, comparing the modifiability of the MIKAEL information architecture with architectures from the state of the art in courseware validation and when required courseware authoring.

We have elicited the following software maintenance scenarios, found in courseware validation:

- The native domain knowledge specification that is used to create the CAVIAr domain model changes.
- There is a change to the courseware specification being used in validation.
- The validation criteria must be changed.

We look at each of these scenarios, in the following subsections.

9.6.1 Native Domain Knowledge Specification Change

Here, we consider the scenario where the native domain model specification being used to bootstrap the CAVIAr Domain Model definition is changed.

Much of the state of the art in courseware authoring limits the authoring effort required for courseware construction through the integration of existing domain knowledge specifications. Section 3.2.3 describes courseware construction tools that allow the use of existing domain knowledge structures, in the form of Semantic Web ontologies, to bootstrap courseware construction. These are as follows:

- VOAT allows for the use of an RDF(S) ontology (described in section 3.2.3.3).
- OntAWare allows for the use of an OWL ontology(described in section 3.2.3.2).
- TANGRAM allows for the use of a SKOS ontology (described in section 3.2.3.1).

MIKAEL demos the use of existing knowledge structures to bootstrap the CAVIAr Domain Model definition (see section 8.8.1) but unlike the tools mentioned above, MIKAEL is not based on one ontology format, but built to import a wide range of knowledge structure definitions to be used in defining the CAVIAr Domain Model. A model transformation mapping from the external knowledge structure specification to the CAVIAr Domain Model is defined for this purpose, as outlined in section 8.8.

Should users of the state of the art wish to use a different domain knowledge specification than the one it is implemented on, it would require major changes to the tool's design. CAVIAr interoperability is based on the software engineering principle of "separation of concerns", where the native knowledge infrastructure concern in courseware construction have been separated. The concern is represented in the model transformation mappings, separate to the programming logic. Should the knowledge infrastructure change or a new one be defined, a new mapping can easily be specified allowing for its integration with the CAVIAr implementation.

9.6.2 Courseware Specification Change

Courseware specifications, such as SCORM and IMS LD discussed in section 2.2.6, allow for interoperability between TEL tools. This interoperability also applies to courseware validation. Specifications allow for validation integration with courseware construction.

Some courseware authoring tools are designed around courseware specifications, these are outlined in section 3.2.1, while other courseware authoring tools just use the courseware specification for interoperability with delivery systems, such as the ACCT, outlined in section 3.2.5.1, and OntAWare, outlined in section 3.2.3.2. Tools that are designed around a courseware specification require considerable effort to allow for export to another specification. Also, if the specification on which a tool is based is updated a major rework would be required for these tools to be compatible with the update.

In section 8.8.4, we have looked at how courseware construction tools, based on CAVIAr, can export specification-compliant courseware through model transformation technology. To do this a model transformation mapping is defined from the CAVIAr Courseware Model specification to the desired specification. A transformation framework then executes the mapping for the given Courseware Model generating the specification-compliant courseware. Integrating changes to courseware specifications requires only updating the transformation mappings, while the introduction of new specifications requires the definition of a new transformation mapping to that specification. This allows for minimal effort to add or update a courseware specification that is used in MIKAEL courseware construction, or a courseware specification needed for validation. This is achieved by isolating the courseware specification concern through model transformation technology.

9.6.3 Validation Criteria Change

One of the key areas of modifiability in courseware validation tools is the ability to define what is valid and invalid. The CAVIAr Validation Model defines the validation criteria using an OCL-defined Validation Model, allowing validation concerns to be separated from the other courseware construction concerns. The Validation Model is not embedded in the programming logic of the tool and can be edited by the course creator. This makes the validation criteria easily modifiable by the course creator, allowing the course creator to define what is valid and invalid for a given courseware.

In section 3.3 we outlined the state of the art in courseware validation. We compare each validation approach with CAVIAr in terms of validation criteria modifiability:

- CoCoA (section 3.3.1) - Validation criteria are defined in its programming logic and cannot be changed. Using CoCoA the course creator cannot define for themselves what is valid and invalid for a given courseware definition. This is one of the main criticisms from course creators who used CoCoA [Brusilovsky and Vassileva, 2003].

- Logic-based Approach (section 3.3.2) - Validation criteria is not accessible to the course creator. The course creator can only have a minimal influence on validation by defining sequencing constraints at the conceptual level and at the course component level.

- IMS Simple Sequencing Trap Detection (section 3.3.3) - It is not possible for the course creator to edit the trap detection validation algorithm defining what is valid and invalid.

- IMS LD “Guidelines” (section 3.3.4) - Guidelines can be defined using the Semantic Web Rule Language (SWRL) [Sicilia, 2007]. Guidelines are defined as a separate concern. It is therefore possible for the the course creator to edit IMS LD guidelines.

The IMS LD guidelines is the only approach from the state of the art that has the capacity of separating the validation concern, allowing the validation criteria to be modified by the course creator. We distinguish our research from IMS LD guidelines in two ways:

- Technical Space - CAVIAr uses the metamodelling technical space whereas IMS LD guidelines are defined in the ontological technical space.

- Scope - Guidelines define good practice in defining IMS LD only. CAVIAr is specification agnostic and a more generic approach to defining what is valid and invalid in courseware.

9.7 Performance

In this section we compare our approach to courseware validation with those used in the state of the art, described in section 3.3, in terms of performance. To do this we look to compare the time complexity of simulation-based validation, as used in the state of the art, with the constraint-based approach, used in CAVIAr. Our aim here is to establish if our approach outperforms the state of the art, particularly with regard to personalised courseware.

Logic-based validation (section 3.3.2) and CoCoA validation (section 3.3.1) are based on simulating a learner's progression through courseware. The processing time of these approaches depend on the number of independent paths through the courseware. The number of independent paths in personalised courseware can be very large. For this reason the CoCoA courseware validation tool only allows for the validation of linear courseware with no branching points. Logic-based validation uses temporal projection to evaluate the potential learner paths through a given courseware. Temporal projection problems are generally accepted to be NP-Complete, but can be solved in polynomial time in a small state space and when the state space is structured [Lin and Dean, 1996], but our research focuses on is courseware that is large and generally has little structure.

The IMS Simple Sequencing trap detection, defined by Lin & Shih (see section 3.3.4), checks for sequencing traps in an IMS Simple Sequencing specification through the traversal of sequencing trees represented using petri nets. To evaluate performance, not only does the traversal of the petri net have to be considered but also the generation of the petri net representation from the IMS Simple Sequencing representation. We also note that problem space for this approach is limited, as it only covers sequencing problems as defined in an IMS Simple Sequencing specification.

In section 3.3.3, we outlined an approach to defining "guidelines" for IMS LD definition. This approach is similar to our approach in that it is constraints driven. Milanović et al. outline how SWRL, the rule language used to define IMS LD "guidelines", can be mapped to OCL. For this reason, we have not included it in our performance comparison [Milanović et al., 2006].

Our OCL-based approach validates courseware in terms of its compositional structure rather than the possible learning paths it represents. This means that increasing the personalisation in a given courseware will only have a limited affect on OCL validation while it would be extremely costly in a logic-based or petri-net approach. The processing time of each CAVIAr Validation Model constraint is dependent on the number of model elements in the CAVIAr models and the number of model elements used in defining each constraint. In [Chimiak-Opoka et al., 2008] the authors report on a set of experiments to compare the performance of Prolog and OCL. Their experiments are conducted on queries that range in complexity and structure. The experiments use Eclipse OCL Validation as an OCL checker. These experiments found that OCL queries can be evaluated in good time, with most evaluated in linear time. Some OCL constructs have been shown to cause performance problems in certain contexts, but good design practice can be used to limit these problems [Cuadrado et al., 2008].

A problem with OCL is that it does not consider aspects such as consequential errors, or the possibility of a single courseware problem causing multiple invariant failures. These problems are discussed by Cabot and Teniente in [Cabot and Teniente, 2006], where they survey popular constraint tools used in Model Driven Architecture (MDA). They note inefficiencies, where constraint checkers do not assess model constraints logically but instead use a brute-force checking method where all constraints are checked against all of the instance model regardless of what invariants have failed already.

9.8 Chapter Conclusion

The aim in this chapter was to evaluate CAVIAr in terms of user acceptance, where user acceptance looks at CAVIAr effectiveness, its cost effectiveness, CAVIAr and MIKAEL usability, CAVIAr modifiability and CAVIAr validation performance. To do this we used three evaluation strategies; an analytical study, an empirical study and a comparative study. The analytical study evaluated each of the CAVIAr metamodel's effectiveness, the empirical study evaluated MIKAEL usability, validation cost effectiveness and its perceived effectiveness. The comparative study compared CAVIAr in terms of its modifiability and validation performance with the state of the art.

In terms of the CAVIAr metamodel's effectiveness, although we found the Domain Model and the Courseware Model not to be sound and the Learning Context Model to be incomplete, there were only minor inconsistencies in the mappings from the CAVIAr definition to the chosen domain conceptualisation. The Domain Model can be mapped to all the main elements of SKOS and is sufficient for its purpose in CAVIAr. There are also only minor differences in the Learning Context Model and the LAOS goal and constraint model. There are a number of IMS LD constructs that cannot be represented in the CAVIAr Courseware Model. IMS LD is quite an ambitious language, which embeds courseware requirements into the courseware definition. The CAVIAr Courseware Model does not attempt to represent the courseware requirements. This is the responsibility of the Learning Context Model and, as already noted, some of the requirements data in this model could be mapped to the IMS LD specification. A minimal level of IMS LD constructs can be mapped to the CAVIAr Courseware Model. We therefore determined the CAVIAr Courseware Model sufficient to allow for validation.

After validation, the course creator had confidence in the courseware produced. This shows that the course creator perceives validation as an effective approach to finding problems in courseware.

The MIKAEL user trials demonstrated a clear positive reaction from course creators in terms of CAVIAr usability. Participants were also positive with respect to CAVIAr tool support provided by MIKAEL and very positive about courseware validation in terms of its usability and the potential time and/or cost savings in course construction.

We compared the state of the art with our approach in two ways, modifiability and performance. We established, our approach is better suited to personalised courseware, due to the expected performance gain that compositional-based validation brings over traversal-based validation. We established that courseware construction and validation approaches based on CAVIAr could be easily modified using model transformation technology to meet the needs of the course creator. We found that modifiability concerns are largely not separated in the state of the art in CAVIAr meaning it requires considerable effort to modify the state of the art validation tools to the needs of a course creator.

Chapter 10

Conclusions

10.1 Introduction

Over the course of this thesis we have presented our constraint-based approach to courseware validation. Our approach allows the course creator to model the courseware construction concerns and validate a given courseware in terms of these concerns using a courseware composition-based model constraint language. We have validated our research by designing and implementing MIKAEL, a courseware construction and validation toolkit based on CAVIAr, our courseware validation Domain Specific Modelling Language (DSML).

Our research investigated how courseware could be validated at the pre-delivery stage of courseware construction. It centred on the explicit representation of courseware requirements, requirements that are generally implicitly held by the course creator. This representation was then used to validate courseware. As mentioned in chapter 1, validation is not a replacement for formative evaluation, but an approach to check that courseware satisfies explicit pre-delivery requirements, defined by the course creator. As validation is a courseware construction activity, integration with the state of the art in courseware construction, was a key research challenge that we addressed. The feasibility of our research was demonstrated through the implementation of a courseware construction software tool that allowed for the explicit representation of courseware requirements and allowed for the validation of courseware based on these requirements.

In this chapter, we summarise our research in section 10.2, outlining how our research has addressed each of the research challenges in section 1.2. This is followed by a discussion on the achievements and contribution of our research, in section 10.3. Finally, in section 10.4, we conclude this chapter by discussing possible future work to extend the research documented in this thesis.

10.2 Research Summary

Courseware validation, in the context of our research, looks at what can be validated in courseware at the pre-delivery/post-construction stage of the courseware life-cycle. Our research is based on using the data available at this stage of the courseware life-cycle, known as the courseware construction concerns, for validation. We used the courseware construction concerns to define a validation framework, outlining the scope of validation.

We have defined a Domain Specific Modelling Language (DSML), which is comprised of a set of metamodels, known as the Courseware Authoring Validation Information Architecture (CAVIAr), that can be used to capture courseware requirements, a courseware definition and details of the learning content used in the courseware. Courseware requirements are principally defined in the Learning Context Model. The Learning Context Model captures the course scope and conceptual instructional constraints in terms of a subject Domain Model. The Courseware Model captures the structure of courseware and includes references to LOs used in the courseware. LOs in the Learning Resource Model have metadata that includes references to Domain Model concept(s). This establishes a relationship between the Courseware Model and the Learning Context Model. Each of the metamodels is defined in terms of its abstract syntax and mapped to a semantic domain. A candidate concrete syntax has been defined for the Domain Model, Learning Context Model and the Courseware Model. To promote interoperability high-level mappings are defined from the CAVIAr metamodels to relevant specifications and standards. We have evaluated each metamodel by comparing it with an established domain conceptualisation for the data the model captures. We found that each of the metamodels fulfilled the requirements of courseware validation.

Our novel composition-oriented approach to courseware validation has been documented in this thesis. We have considered three major validation categories using CAVIAr, courseware validation pre-requisites, courseware model validation and learning context validation. Courseware validation pre-requisites defines data in CAVIAr models required for validation to take place, such as essential LO metadata categories. Courseware model validation defines constraints on the Courseware Model in isolation of the learning context, while learning context validation validates courseware using the courseware construction concerns defined in the Learning Context Model. We described each of these constraint categories in detail and exemplified them using the Object Constraint Language (OCL). The OCL constraints were defined in the context of the CAVIAr metamodels and constrained the allowable Courseware Model definitions. We have compared our approach with the state of the art, which validates courseware by primarily simulating learner progression through courseware. We found that as courseware becomes more adaptive, the simulation approach will result in complexity problems, whereas our approach is more suited to validation of adaptive courseware. We have also found that defining the Validation Model in a separate OCL file, which can be modified by the course creator, empowers the course creator to define what is valid and is not valid in the courseware they have constructed.

We have also outlined a courseware validation process, outlining how a course creator validates courseware using CAVIAr. This process essentially outlines how the course creator should define each CAVIAr model to allow for validation. We have exemplified each of the activities described in the courseware validation process using a case-study application of the validation process, the validation of DCU module CA218 - “Introduction to Databases”.

The feasibility of our validation approach has been addressed through the development of a proof of concept implementation known as the Management Infrastructure for Knowledge-based Adaptive E-Learning (MIKAEL). MIKAEL is a courseware construction and validation software tool that is based on CAVIAr. MIKAEL allows the course creator to intuitively create a CAVIAr Learning Context Model to represent the courseware construction concerns. The courseware construction concerns defined using the CAVIAr Learning Context Model are then used to generate a Courseware Model. Generation is performed using a model transformation, which generates a Courseware Model based on the courseware construction concerns defined in the Learning Context Model. The course creator can define various mappings to generate different types of Courseware Models. The course creator can use the mappings to specify what instructional design he or she wishes to use in courseware. We have also outlined how model transformation technology can be used to provide for CAVIAr interoperability with TEL and related specifications in the MIKAEL tool. This allows the course creator to generate many of the CAVIAr models, limiting the effort involved in courseware construction and validation. Model transformation technology also allows for the course creator to export a Courseware Model to a TEL specification.

MIKAEL was used in a series of user trials that examined CAVIAr-based courseware construction in terms of its usability and benefits to the course creator. We recieved positive results from the users of the system. Of particular note were the positive results regarding courseware validation, where course creators stated they would be more confident in a constructed courseware post-validation.

10.3 Discussion

Our approach to courseware validation is novel, in that it does not simulate learner interaction with the courseware, but instead looks to validate the courseware based on its structure and composition. Validation is achieved by allowing the explicit representation of courseware and the courseware construction concerns using a DSML known as CAVIAr. Validation criteria are expressed as a set of constraints that constrain what is allowable courseware structure. The adaptivity found in personalised courseware greatly adds to the complexity of the state of the art courseware validation approaches that are based on learner simulation, we have established that our constraint-based approach is better suited to validation of adaptive courseware when compared with the state of the art.

One of the principle problems found in the state of the art courseware validation tool support is that the definition of what is valid and invalid in courseware is hidden from the course creator in the tool's programming logic [Brusilovsky and Vassileva, 2003]. Our approach exposes the validation definition to the course creator, empowering the course creator to define what he or she deems as valid or invalid. This flexibility is a key contribution of our work.

Using the metamodelling technical space allows us to utilise model transformation technology for converting to, from and between CAVIAr models. We utilise model transformation in three key areas:

- Importing a Domain Model - Importing a Domain Model from a knowledge structure specification.
- Importing a Courseware Specification - Importing a Courseware Model from a TEL or AEH specification.
- Exporting a Courseware Model - Exporting courseware defined as a CAVIAr Courseware Model into a TEL specification.

The import and export of models illustrates our interoperability approach with TEL tool support, allowing for courseware validation to be integrated with courseware construction. Model transformation technology allows us to bootstrap a Domain Model definition using a knowledge structure from any source as long as an abstract syntax is explicitly defined for the data source and it can be mapped to the CAVIAr Domain Model definition. Courseware specifications can also be used to define CAVIAr models using model transformation technology. We can also use model transformation technology to export a Courseware Model to any specification where an abstract syntax is explicitly defined for the specification, and a mapping is defined from the CAVIAr Courseware Model to the specification. This provides for maintainable interoperability, where if an import or export specification changes all that is required to remain interoperable is an altering of the mapping. Further still, if the course creator wishes to interoperate with a specification which was unforeseen when CAVIAr tool support was developed it can be easily integrated by just defining a new mapping. This allows CAVIAr validation to be easily integrated into existing courseware construction tools and methodologies.

As courseware gets more complex, through personalisation, and as the course creator becomes more abstracted from the construction of courseware, through reuse and collaborative authoring, it will become a necessity that the course creation tool can assist the course creator in verifying that the courseware created satisfies the course creator's requirements. Although it is not possible to validate issues such as learner motivation and the effects of instructional material until a formative evaluation has taken place, we have found that validation can test the courseware for a wide range of pedagogical and non-pedagogical problems that can be defined in terms of the courseware structure and the courseware construction concerns. We have also found that these types of problems will get more difficult to check for, as courseware becomes increasingly adaptive. Validation will also make formative evaluation more powerful, as the more mundane courseware problems can be checked automatically and will not get in the way of evaluating more complex pedagogical aspects of courseware such as those evaluated in a courseware's formative evaluation.

Another point of interest raised during the user trials, is that courses are a necessity in the corporate world, indeed the delivery of courses with specific learning outcomes are part of compliance regulations where organisations are legally obligated to train their staff on certain issues. Courseware validation allows for the verification that a given courseware is compliant with defined legislative requirements that it has been designed to address.

10.4 Future Work

A number of enhancements to our validation approach have been identified as future work through our own insight and through feedback from course creators during the MIKAEL user trials. These enhancements are as follows:

- Intuitive Validation Model definition.
- Enhancements to the Courseware Model.

- Categorisation of courseware problems.
- Correction integration.

10.4.1 Intuitive Validation Model Definition

The Validation Model is defined using OCL. OCL is a language designed for software engineers to define constraints on UML models to remove ambiguity (as outlined in section 2.4). The principle user of CAVIAr is the course creator. We cannot expect the course creator to be able to define the CAVIAr Validation Model in OCL. In section 6.2.2, we described one approach that we have investigated to allow for a more intuitive interface for defining the CAVIAr Validation Model, the application of a DSML to abstract the complexity in defining OCL constraints. OCL is generated from a DSML definition. Using this approach a Validation Model is defined using the DSML and OCL is then generated from the DSML definition. This provides a powerful interface for the course creator to define a Validation Model. We have presented an experimental proof of concept to this effect, in section 6.2.2. Once integrated it will provide course creators with an intuitive way to define a CAVIAr Validation Model.

Another proposal to allow for a more intuitive approach to defining a Validation Model is to apply the reuse and componentisation paradigm, used for LOs [Wiley, 2001], to validation rules. In this approach, Learning Object Repositories (LORs) store validation rules. These validation rules are formalised in OCL and also annotated using metadata that the course creator understands. The course creator can search for validation rules in the LOR based on the validation needs of the course creator. The course creator can make a decision on whether or not to use a validation rule based on the constraint's annotation. Validation rules, found in the LOR, can also be aggregated together. Various validation rules aggregated together can be used to ensure an instructional design theory has been applied correctly to courseware. This aggregation of courseware validation constraints can also be annotated and stored in a LOR, therefore allowing for the reuse of a Validation Model that ensures the correct application of an instructional design theory.

10.4.2 Enhancement of the Courseware Model

In section 9.5.1.4, we evaluated the CAVIAr Courseware Model by comparing it with the IMS LD specification. We found that some elements of the IMS LD specification could not be represented in the CAVIAr Courseware Model. This is due to the ambitious nature of the IMS LD specification that seeks to fully describe the whole teaching process [Koper and Olivier, 2004]:

- Defining staff roles as well as learner roles.
- Defining support activities as well as the main learning activities.
- Defining both single and multiple user models.
- Allowing for blended and online only learning

A future enhancement to the CAVIAr Courseware Model would be to incorporate these aspects of the IMS LD specification that are currently lacking.

Another enhancement to the CAVIAr Courseware Model would be the definition of an intuitive flow-based concrete syntax. In section 3.2.5, we outlined approaches from the state of the art that model a "unit of learning" using a flow-oriented notation. One of our criticisms of flow-oriented approaches to courseware representation is that flow-oriented notation can get very verbose when modelling highly adaptive courseware, and in courseware where the learner has a lot of free choice. It would be interesting to investigate approaches to representing the courseware using flow-oriented notation, such as:

- Investigating new business flow notations, such as the Business Process Modelling Notation (BPMN) [White, 2004], rather than software engineering oriented notation, as a more intuitive way of representing courseware flow.
- Generating a flow-notation for each individual learner stereotype to limit variability in one diagram.

10.4.3 Categorisation of Courseware Problems

One of the problems with OCL is that constraints have no semantics associated with them. Some attempts to overcome this can be found in the EMF validation project, which allows for the mapping of various severity levels to different OCL constraints [Steinburg et al., 2008]. We found during the course of the MIKAEL user trials that course creators desire categorisation of the validation errors found in courseware to help them decide whether a validation error is something that can be ignored or something that must be addressed.

To do this a metadata layer could be defined on OCL constraints where each constraint is annotated to indicate the severity of the problem and other such data that is useful to the course creator. As the OCL parser used in MIKAEL is open source, we could extend this OCL parser to parse this metadata, allowing for the severity levels to be incorporated into validation.

10.4.4 Correction integration

Integrating correction support into the CAVIAr validation framework is a natural progression. Validation data could be used to recommend possible correction strategies to the course creator. The category of a courseware problem identified as outlined in section 10.4.3 could also be used as a basis for recommending a correction strategy. Indeed for each category of validation error defined in the CAVIAr Validation Model, there could be a corresponding correction strategy, which is initialised by the particulars of the constraint that fails during validation.

Bibliography

[ACM, 1998] ACM (1998). The ACM computing classification system. Technical report, Association of Computer Machinery.

[ADL, 2004] ADL, A. (2004). SCORM 2004 Overview. Available from: http://www.adlnet.gov/scorm/index.cfm.

[Albert and Stefanutti, 2003] Albert, D. and Stefanutti, L. (2003). Knowledge structures and didactic model selection in learning object navigation. In *Proccedings of the Joint Workshop in Cognition and Leanring through Media-Communicaiton for Advanced E-Leanring (JWCL)*, pages 1–10, Berlin, Germany.

[Altova, 2005] Altova (2005). *Altova Xmlspy 2005 User & Reference Manual*. Vervante.

[Aroya et al., 2002] Aroya, L., Cristea, A. I., and Dicheva, D. (2002). A Layered Approach towards Domain Authoring. In *Proceedings of The International Conference on Artificial Intelligence (ICAI02)*, pages 615–621. CSREA.

[Arthorne and Laffra, 2004] Arthorne, J. and Laffra, C. (2004). *Official Eclipse 3.0*. Number 0321268385 in Eclipse Series. Addison Wesley.

[ATLAS Group, 2006] ATLAS Group (2006). Atl: Atlas transformation language - atl user manual. Technical report, LINA & INRIA.

[Bajnai and Stienberger, 2003] Bajnai, J. and Stienberger, C. (2003). Eduweaver the web-based courseware design tool. In *Proceedings of IADIS International Conference Internet/WWW 2003*, Algarve, Portugal. IADIS.

[Baldoni et al., 2006] Baldoni, M., Baroglio, C., Martelli, A., Patti, V., and Torasso, L. (2006). Verifying the compliance of personalized curricula to curricula models in the semantic web. In *Proceeding of the Semantic Web Personalization Workshop at the Third European Semantic Web Conference (ESWC2006)*. Springer-Verlag LNCS Series.

[Baldoni et al., 2004a] Baldoni, M., Baroglio, C., and Patti, V. (2004a). Web-Based Adaptive Tutoring: An Approach Based on Logic Agents and Reasoning about Actions. *Artificial Intelligence Review*, 22(1):3–39.

[Baldoni et al., 2004b] Baldoni, M., Baroglio, C., Patti, V., and Torasso, L. (2004b). Reasoning about learning object metadata for adapting SCORM courseware. In *Proceeding of the International Workshop on Engineering the Adaptive Web: Methods and Technologies for personalization adn adaptation in the Semantic Web (EAW204)*, pages 4–13. Springer-Verlag LNCS Series.

[Baldoni et al., 2004c] Baldoni, M., Giordano, L., Martelli, A., and Patti, V. (2004c). Programming Rational Agents in a Modal Action Logic. *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*, 41(2-4):207–257.

[Beckert et al., 2002] Beckert, B., Keller, U., and Schmitt, P. (2002). Translating the object constraint language into first-order predicate logic. In *In Proceedings, VERIFY, Workshop at Federated Logic Conferences (FLoC)*.

[Bengtsson et al., 2004] Bengtsson, P., Lassing, N., Bosch, J., and van Vliet, H. (2004). Architecture-level modifiability analysis (alma). *The Journal of Systems and Software*, 69:129–147.

[Bézivin et al., 2005] Bézivin, J., Jouault, F., Rosenthal, P., and Valduriez, P. (2005). *Model Driven Architecture*, chapter Modeling in the Large and Modeling in the Small, pages 33–46. Springer.

[Blackerby et al., 2002] Blackerby, C., Shelton, C., and Gillis, L. B. (2002). A report to the 78th texas legislature on investigating quality of online courses. Technical report, Texas Education Agency.

[Briggs et al., 1991] Briggs, L., Gustafson, K. L., and Tillman, M. H., editors (1991). *Instructional Design: Principles and Applications*. Educational Technology Publications, New Jersey, USA.

[Brusilovsky, 1996] Brusilovsky, P. (1996). Methods and techniques of adaptive hypermedia. *Methods and Techniques of Adaptive Hypermedia*, 6(2-3):87–129.

[Brusilovsky, 2000] Brusilovsky, P. (2000). Concept-based courseware engineering for large scale web-based education. In Davies, G. and Owen, C., editors, *Proceedings of WebNet2000, World Conference of the WWW and Internet*, pages 69–74, San Antonio, TX. AACE.

[Brusilovsky et al., 1998] Brusilovsky, P., Eklund, J., and Schwarz, E. (1998). Web-based education for all: a tool for development adaptive courseware. *Computer Networks and ISDN Systems*, 30(1–7):291–300.

[Brusilovsky and Vassileva, 2003] Brusilovsky, P. and Vassileva, J. (2003). Course sequencing techniques for large-scale web-based education. *International Journal Continuing Engineering Education and Lifelong Learning*, 13(1/2):75–94.

[Budd, 2004] Budd, J. W. (2004). Mind maps as classroom exercises. *Journal of Economic Education*, 35(1):35–46.

[Cabot and Teniente, 2006] Cabot, J. and Teniente, E. (2006). Constraint support in mda tools: A survey. In *Model Driven Architecture ? Foundations and Applications*, number 4066 in LNCS, pages 256–267. Springer-Verlag.

[Carey and Dick, 1991] Carey, L. M. and Dick, W. (1991). *Instructional Design: Principles and Applications*, chapter Summative Evaluation, pages 269–311. Educational Technology Publications, 2nd edition.

[Chimiak-Opoka et al., 2008] Chimiak-Opoka, J., Felderer, M., Lenz, C., and Lange, C. (2008). Querying uml models using ocl and prolog: A performance study. In *Proceeding of the Software Testing Verification and Validation Workshop, 2008. ICSTW '08*, pages 81–88. IEEE.

[Cristea and Aroya, 2002] Cristea, A. and Aroya, L. (2002). Adaptive authoring of adaptive educational hypermedia. In DeBra, P., Brusilovsky, P., and Conejo, R., editors, *Proceedings of the 2nd International Conference on Adaptive Hypermedia (AH2002)*, pages 122–132, Malaga, Spain. Springer-Verlag.

[Cristea et al., 2007] Cristea, A., Smits, D., and deBra, P. (2007). Towards a generic adaptive hypermedia platform: a conversion case study. *Journal of Digital Information*, 8(3).

[Cristea and de Mooij, 2003] Cristea, A. I. and de Mooij, A. (2003). LAOS: Layered WWW AHS Authoring Model and their corresponding Algebraic Operators. In *Proceedings of The Twelfth International World Wide Web Conference (WWW03), Alternate Track on Education*. ACM.

[Cristea et al., 2003a] Cristea, A. I., Smits, D., and de Bra, P. (2003a). Writing MOT, Reading AHA! - converting between an authoring and a delivery system for adaptive educational hypermedia. In *Proceedings of The Third International Workshop on Authoring of Adaptive and Adaptable Educational Hypermedia at AIED05*.

[Cristea et al., 2003b] Cristea, A. I., Stewart, C., Brailsford, T., and Cristea, P. (2003b). Evaluation of Interoperability of Adaptive Hypermedia Systems: testing the MOT to WHURLE conversion in a classroom setting. In *Proceedings of The Third International Workshop on Authoring of Adaptive and Adaptable Educational Hypermedia at AIED05*.

[Cuadrado et al., 2008] Cuadrado, J. S., Jouault, F., Molina, J. G., and Bézivin, J. (2008). Optimization patterns for OCL-based model transformation. In *Proceedings of the 8th OCL Workshop at the UML/MoDELS Conferences*, Toulouse, France.

[Czarnecki and Eisenecker, 2000] Czarnecki, K. and Eisenecker, U. (2000). *Generative Programming: Methods, Tools and Applications*. Addison-Wesley Professional.

[Czarnecki and Helson, 2006] Czarnecki, K. and Helson, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645.

[Daconta et al., 2003] Daconta, M. C., Obrst, L. J., and Smith, K. T. (2003). *The Semantic Web: A Guide to the Future of XML, Web Services and Knowledge Management* . Wiley Publications, Indianapolis, Indiana.

[Dagger, 2006a] Dagger, D. (2006a). Authoring standards based personalised elearning. In Reeves, T. and Yamashita, S., editors, *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, pages 2680–2685, Chesapeake, VA. AACE.

[Dagger, 2006b] Dagger, D. (2006b). *Personalised eLearning Development Environments*. PhD thesis, University of Dublin.

[Dagger et al., 2003] Dagger, D., Conlan, O., and Wade, V. P. (2003). An architecture for candidacy in adaptive elearning systems to faciltate the reuse of learning resources. In *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, page 49, Chesapeake, VA. AACE.

[Dawson and Howes, 1998] Dawson, F. and Howes, T. (1998). vCard MIME Directory Profile. RFC Editor.

[DCMI, 2006] DCMI (2006). Dublin core metadata. http://dublincore.org.

[DeBra and Calvi, 1998] DeBra, P. and Calvi, L. (1998). AHA!: a Generic Adaptive Hypermedia System. In *Proceedings of the 2nd Workshop on Adaptive Hypertext and Hypermedia*, http://wwwis.win.tue.nl/ah98/Proceedings.html. Eindhoven University of Technology.

[DeBra et al., 1999] DeBra, P., Houben, G.-J., and Wu, H. (1999). AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. In *Proceedings of the 10th ACM Conference on Hypertext and Hypermedia*, pages 147–156. ACM.

[Dick and Carey, 1991] Dick, W. and Carey, L. M. (1991). *Instructional Design: Principles and Applications*, chapter Formative Evaluation, pages 227–267. Educational Technology Publications, 2nd edition.

[Djurić et al., 2006] Djurić, D., Gaševic, D., and Devedžic, V. (2006). The Tao of Modeling Spaces. *Journal of Object Technology*, 6(x). forthcoming.

[Doniger et al., 2003] Doniger, S. W., Salomonis, N., Dahlquist, K. D., Vranizan, K., Lawlor, S. C., and Conklin, B. R. (2003). MAPP:Finder: using Gene Ontology and GenMAPP to create a global gene-expression profile from micro arraydata. *Genome Biology*, 4(R7).

[DresdenOCL, 2007] DresdenOCL, T. (2007). Dresden OCL Toolkit. http://dresden-ocl.sourceforge.net/.

[Dwolatzky et al., 2002] Dwolatzky, B., Kennedy, I., and Owens, J. (3-4 Jan. 2002). Modern software engineering methods for developing courseware. *Engineering Education 2002: Professional Engineering Scenarios (Ref. No. 2002/056), IEE*, 2:–32/6.

[Eclipse Foundation, 2008] Eclipse Foundation (2008). Eclipse M2M Project . http://www.eclipse.org/m2m.

[Eclipse Foundation, 2009] Eclipse Foundation (2009). Eclipse M2T Project. http://www.eclipse.org/modeling/m2t/.

[Eclipse MDT, 2008] Eclipse MDT (2008). Model development tools (mdt).

[Eifel, 2007] Eifel (2007). XML SCORM Studio. http://www.eife-l.org/publications/softwarecenter/xmlscormstudio.

[Eklund and Brusilovsky, 1999] Eklund, J. and Brusilovsky, P. (1999). InterBook: An Adaptive Tutoring System. *UniServe Science News*, 12:8–13.

[Eriksson et al., 2003] Eriksson, H.-E., Penker, M., Lyons, B., and Fado, D. (2003). *UML 2.0 Toolkit*. Wiley Publications, Indianapolis, Indiana.

[EU Bologna Agreement, 2000] EU Bologna Agreement (2000). The Bologna Declaration on the European space for Higher Education: an explanation. Technical report, Confederation of EU Rectors' Conferences and the Association of European Universities.

[FOKUS, Fraunhofer Institute, 2006] FOKUS, Fraunhofer Institute (2006). Open Source Library for OCL (OSLO). http://oslo-project.berlios.de/.

[Frankel, 2003] Frankel, D. S. (2003). *Model Driven Architecture*. Wiley Publications, Indianapolis, Indiana.

[Gagné et al., 2005] Gagné, R., Wager, W., Golas, K., and Keller, J. (2005). *Principles of Instructional Design*. Wadsworth, California, USA, 5th edition.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: elements of reusable object-oriented software*. Addison Wesley, New Jersey, USA.

[Gašević et al., 2006] Gašević, D., Djurić, D., and Devedžić, V. (2006). *Model-Driven Architecture and Ontology Development*, chapter The Ontology Definition Metamodel (ODM), pages 181–199. Springer-Verlag.

[Gomez-Perez et al., 2004] Gomez-Perez, A., Corcho, O., and Fernandez-Lopez, M. (2004). *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer.

[Griffiths et al., 2009] Griffiths, D., Beauvoir, P., Liber, O., and Barrett-Baxendale, M. (2009). From reload to recourse: learning from ims learning design implementations. *Distance Education*, 30(2):201 – 222.

[Gronback, 2009] Gronback, R. C. (2009). *Eclipse Modeling Project: A Domain-Specific Language Toolkit*. Addison Wesley Professional, pre-print edition.

[Grützner et al., 2004] Grützner, I., Weibelzahl, S., and Waterson, P. (2004). Improving courseware quality through lifecycle encompassing quality assurance. In *Proceeding of the Symposium on Applied Computing (SAC '04)*, Nicosia, Cyprus. ACM Press.

[Guizzardi et al., 2005] Guizzardi, G., Pires, L. F., and van Sinderen, M. (2005). Ontology-based evaluation and design of domain-specific visual modeling languages. In *Proceedings of the 14th International Conference on Information Systems Development*, number 3713 in LNCS, pages 691–705.

[Harel and Rumpe, 2004] Harel, D. and Rumpe, B. (2004). Meaningful modeling: What's the semantics of "semantics". *Computer*, 37(10):64–72.

[Hemenway and Calishain, 2003] Hemenway, K. and Calishain, T. (2003). *Spidering Hacks*. Number 0596005776. O'Reilly Publications, Sepastopol, CA, USA.

[Hendler, 2008] Hendler, J. (2008). Web 3.0: Chicken farms on the semantic web. *IEEE Computer*, 41(1):106–108.

[Holohan, 2003] Holohan, E. (2003). Automating the Generation of Courseware. Master's thesis, Dublin City University.

[Holohan et al., 2005] Holohan, E., McMullen, D., Melia, M., and Pahl, C. (2005). Adaptive Courseware Generation based on Semantic Web Technologies. In *Proceeding of the International Workshop on Applications of Semantic Web Technologies for E-Learning (SW-EL2005) at the Twelveth International Conference on Artificial Intelligence in Education (AIED2005)* . IOS Press.

[Holohan et al., 2006] Holohan, E., McMullen, D., Melia, M., and Pahl, C. (2006). Adaptive Courseware Generation based on Semantic Web Technologies. In *Proceeding of the Sixth International Conference on Advanced Learning Technologies (ICALT2005)*, pages 967–969. IEEE Computer Society.

[Horrocks et al., 2004] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Technical report.

[Hummel et al., 2004] Hummel, H., Manderveld, J., Tattersall, C., and Koper, R. (2004). Educational modelling language and learning design: new opportunities for instructional reusability and personalised learning. *International Journal of Learning Technology*, 1(1):110–126.

[Hussmann et al., 2000] Hussmann, H., Demuth, B., and Finger, F. (2000). Modular architecture for a toolset supporting ocl. Number 1939 in LNCS, pages 278–293. Springer.

[IEEE LTSC, 2002] IEEE LTSC (2002). LTSC WG12:Learning Object Metadata. IEEE Learning Technology Standards Committee.

[IMS, 2002] IMS (2002). IMS Reusable Definition of Competency or Educational Objective . Technical Report 1.0, IMS Global Learning Consortium.

[IMS, 2003a] IMS (2003a). IMS Content Packaging (Version 1.3.1) Overview. Technical Report 03/19, IMS, Global Learning Consortium. Available from: www.imsglobal.org/content/packaging/.

[IMS, 2003b] IMS (2003b). IMS Learning Design. Technical report, IMS, Global Learning Consortium.

[IMS, 2003c] IMS (2003c). IMS Simple Sequencing Specification. Technical Report 03/19, IMS, Global Learning Consortium. Available from http://www.imsglobal.org/simplesequencing/.

[IMS, 2005] IMS (2005). IMS Learner Information Package. Technical Report Version 1.0.1, IMS Global Learning Consortium.

[Ismail, 2001] Ismail, J. (2001). The design of an e-learning system: Beyond the hype. *The Internet and Higher Education*, 4(3-4):329 – 336.

[Janjua, 2008] Janjua, U. T. (2008). Model Based OCL Generation. MSE, Dublin City University.

[Jouault and Kurtev, 2005] Jouault, F. and Kurtev, I. (2005). Transforming Models with ATL. In *Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005*.

[Jovanović et al., 2006a] Jovanović, J., Gašević, D., and Devedzic, V. (2006a). Dynamic Assembly of Personalized Learning Content on the Semantic Web. In *Proceeding of the 3rd European Semantic Web Conference 2006 (ESWC2006)*. Springer Verlag.

[Jovanović et al., 2006b] Jovanović, J., Gašević, D., and Devedzic, V. (2006b). Ontology-based Automatic Annotation of Learning Content. *International Journal on Semantic Web and Information Systems*, 2(2):91–119.

[Jovanović et al., 2005] Jovanović, J., Gašević, D., Verbert, K., and Erik, D. (2005). Ontology of learning object content structure. In *Proceeding of the 12th International Conference on Artifical Intelligence in Education*, pages 322–329. IOS Press.

[Karampiperis and Sampson, 2004] Karampiperis, P. and Sampson, D. (2004). Adaptive hypermedia authoring: From adaptive navigation to adaptive learning support. In *Proceedings of the 2nd International Workshop on Authoring of Adaptive and Adaptable Hypermedia at AH2004*, Eindhoven, Netherlands.

[Kay, 2000] Kay, J. (2000). Stereotypes, student models and scrutability. In Gauthier, G., Frasson, C., and VanLehn, K., editors, *Proceedings of Intelligent Tutoring Systems 2000 (ITS 2000)*, number 1839 in LNCS, pages 19–30. Springer-Verlag.

[Kenny, 2006] Kenny, C. (2006). Automated tutoring for a databases skills training environment. Master's thesis, Dublin City University.

[Koper, 2005] Koper, R. (2005). *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*, chapter An Introduction to Learning Design, pages 3–19. Springer.

[Koper and Olivier, 2004] Koper, R. and Olivier, B. (2004). Representing the learning design of units of learning. *Educational Technology & Society*, 7(3):97–111.

[Kurtev et al., 2006] Kurtev, I., Bézivin, J., Jouault, F., and Valduriez, P. (2006). Model-based DSL Frameworks. In *SSPSLA'06: Companion of the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 602–616, New York, USA. ACM Press.

[Laforcade and Choquet, 2006] Laforcade, P. and Choquet, C. (2006). Next Step for Educational Modeling Languages: The Model Deiven Endineering and Reengineering Approach. In *Proceeding of the Sixth IEEE International Conference on Advanced Learning Technologies (ICALT2006)*, pages 747–746. IEEE Computer Society.

[Lenat, 1996] Lenat, D. (1996). Cyc: a large-scale investement in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38.

[Likert, 1932] Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, (140):1–155.

[Lin and Shih, 2009] Lin, F. H. and Shih, T. K. (2009). Automatic Trap Detection: A Debugging Mechanism for Abnormal Specification in the IMS Sequencing Controls. *IEEE Transactions on Learning Technology*, 1(3):176–189.

[Lin and Dean, 1996] Lin, S. and Dean, T. (1996). Localized temporal reasoning using subgoals and abstract events. *Computational Intelligence*, 12:423–449.

[Martin et al., 2007] Martin, B., Mitrovic, A., and Suraweera, P. (2007). Domain Modelling with Ontology: A Case Study. In *Proceedings of the 5th International Workshop on Authoring for Adaptive and Adaptable Hypermedia (A3H)*, Corfu, Greece.

[Martínez-Ortiz et al., 2007] Martínez-Ortiz, I., Moreno-Ger, P., Sierra, J. L., and Fernández-Manjón, B. (2007). *Computers and Education: E-learning – from theory to practice*, chapter Educational Modeling Languages: A Conceptual Introduction and a High-Level Classification, pages 27–40. Springer.

[Martínez-Ortiz et al., 2009] Martínez-Ortiz, I., Sierra, J. L., and Fernández-Manjón, B. (2009). Translating e-leanring flow-oriented activity sequencing descriptions into rule-based designs. In *Proceedings of the 6th International Conference on Information Technology: New Generations*, Las Vegas, Nevada, USA.

[McMullen, 2007] McMullen, D. (2007). Using ontology technology to support content generation and run time adaptivity in e-learning environments. Master's thesis, Dublin City University.

[McMullen et al., 2005] McMullen, D., Holohan, E., Melia, M., and Pahl, C. (2005). Knowledge-Driven Learning Technology Systems. In *Proceeding of the Sixth Annual Irish Educational Technology User's Conference (EdTech05)*. ILTA.

[Melia et al., 2006] Melia, M., Barrett, R., and Pahl, C. (2006). A Model-Based Approach to SCORM Sequencing. In *Proceeding of the Sixth Annual Irish Educational Technology User's Conference (EdTech06) - Research Track*. ILTA.

[Melia et al., 2005] Melia, M., Holohan, E., McMullen, D., and Pahl, C. (2005). Ontology-based Adaptive Content Navigation. In *Proceeding of the First International Conference on Methods and Technologies for Learning (ICMTL2005)*, pages 435–440. WIT Press.

[Melia and Pahl, 2006a] Melia, M. and Pahl, C. (2006a). Automatic Validation of Learning Object Compositions. In *Proceedings of the Information Technology and Telecommunications Conference IT&T2006*, number 1649-1246, Carlow, Ireland. TecNet.

[Melia and Pahl, 2006b] Melia, M. and Pahl, C. (2006b). Semantically-enabled Model Driven Course Development. In *Proceeding of the First European Conference on Technology Enhanced Learning (EC-TEL06) - Doctoral Consortium Session*. EC-TEL06 Workshop Proceedings.

[Melia and Pahl, 2007a] Melia, M. and Pahl, C. (2007a). An information architecture for courseware validation. In *Proceedings of the 8th Annual Irish Educational Technology User's Conference (EdTech2007)*. ILTA.

[Melia and Pahl, 2007b] Melia, M. and Pahl, C. (2007b). An information architecture for validating courseware. In Massart, D. and Colin, J.-N., editors, *Proceeding of the First International Workshop on Learning Object Discovery and Exchange (LODE2007) at EC-TEL2007*, Crete, Greece. CEUR Workshop Proceedings.

[Melia and Pahl, 2007c] Melia, M. and Pahl, C. (2007c). Pedagogical validation of courseware. In Duval, E., Klamma, R., and Wolpers, M., editors, *Proceedings of the Second European Conference on Technology Enhanced Learning*, number 4753 in LNCS. Springer.

[Melia and Pahl, 2008] Melia, M. and Pahl, C. (2008). Towards the validation of adaptive educational hypermedia using CAVIAr. In *Proceeding of the 6th International Workshop on Authoring Adaptive and Adaptable Hypermedia (A3H2008)*, AH2008 Workshop Proceedings, Hannover, Germany.

[Melia and Pahl, 2009] Melia, M. and Pahl, C. (2009). Constraint-based validation of adaptive e-learning courseware. *IEEE Transactions on Learning Technology*, 2(1):37–49.

[Mendling et al., 2007] Mendling, J., Neumann, G., and van der Aalst, W. (2007). On the correlation between process model metrics and errors. In *ER '07: Tutorials, posters, panels and industrial contributions at the 26th international conference on Conceptual modeling*, number 978-1-920682-64-4, pages 173–178, Darlinghurst, Australia. Australian Computer Society, Inc.

[Mernik et al., 2005] Mernik, M., Anthony, J. H., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computer Survey*, 37(4):316–344.

[Milanović et al., 2006] Milanović, M., Gašević, D., Giurca, A., Wagner, G., and Devedžić, V. (2006). Sharing owl/swrl and uml/ocl rules. In *In proceedings of OCLApps 2006: OCL for (Meta-)Models in Multiple Application Domains*, Genova, Italy.

[Miles and Bechhofer, 2009] Miles, A. and Bechhofer, S. (2009). Skos simple knowledge organization system reference. Candidate recommendation, W3C, http://www.w3.org/TR/2009/CR-skos-reference-20090317/.

[Miles and Brickley, 2005] Miles, A. and Brickley, D. (2005). SKOS Core Guide. Technical report.

[Mississippi State University, 2007] Mississippi State University (2007). InSite Studio. http:thorax.erc.msstate.eduinsitedefault.aspx.

[Mitrovic et al., 2007] Mitrovic, A., Martin, B., and Suraweera, P. (2007). Intelligent Tutors for All: The Constraint-Based Approach. *IEEE Inteligent Systems*, 22(4):38–45.

[Mitrovic et al., 2006] Mitrovic, A., Suraweera, P., Martin, B., Zakharov, K., Milik, N., and Holland, J. (2006). Authoring constraint-based tutors in aspire. In *Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS2006)*, pages 41–50. Springer-Valag.

[Motelet et al., 2007] Motelet, O., Baloian, N., and Pino, J. A. (2007). *Learning Object: Standards, Metadata, Repositories and LCMS*, chapter Learning Object Metadata and Automatic Processes: Issues and Perspectives, pages 185–220. Informing Science.

[Murray, 2003] Murray, T. (2003). *Authoring tools for Advanced Technology Learning Environments*, chapter An Overview of Intelligent Tutoring System Authoring Tool: Updated Analysis of the State of the Art, pages 491–538. Kluwer Academic Publishers.

[Nesbit et al., 2003] Nesbit, J., Belfer, K., and Leacock, T. (2003). Learning object review instrument (lori) - user manual. User Manual 1.5, eLera.

[Niles and Pease, 2001] Niles, I. and Pease, A. (2001). Origins of the standard upper merged ontology: A proposal for the ieee standard upper ontology. In *In Working Notes of the IJCAI-2001 Workshop on the IEEE Standard Upper Ontology*, Seattle Washington.

[NQAI, 2003] NQAI (2003). National quailifications authority of ireland - national framework of quailifications. http://www.nfq.ie/nfq/en/public_resources/documents/TheNFQ-AnOverview.pdf.

[Olivier and Tattersall, 2005] Olivier, B. and Tattersall, C. (2005). *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*, chapter The Learning Design Specification, pages 21–40. Springer, Berlin, Germany.

[OMG, 2003a] OMG (2003a). Meta Object Facility (MOF) 2.0. OMG Final Adopted Specification.

[OMG, 2003b] OMG (2003b). OCL 2.0. OMG Final Adopted Specification.

[OMG, 2005] OMG (2005). Meta Object Facility (MOF) 2.0 Query View Transformation (QVT). OMG Final Adopted Specification.

[OMG, 2007] OMG (2007). Unified modeling language 2.1.2. OMG Final Adopted Specification.

[Padrón et al., 2006] Padrón, C. L., Díaz, P., and Aedo, I. (2006). MD2 Method: The Didactic Materials Creation from a Model Based Perspective. In *Proceeding of the First European Conference on Technology Enhanced Learning (EC-TEL 2006)*. Springer-Verlag LNCS.

[Padrón et al., 2007] Padrón, C. L., Díaz, P., and Aedo, I. (2007). The role of evaluation in an effective development of didactic material: The MD2 approach. In *Proceeding of the Seventh International Conference on Advanced Learning Technologies (ICALT2007)*, Niigata, Japan. IEEE Computer Society.

[Padrón et al., 2008] Padrón, C. L., Zarraonandía, T., Díaz, P., and Aedo, I. (2008). The evaluation within the development and deployment of IMS LD-based didactic materials: THe MD2 + runtime adaptation approach. In *Proccedings of the First Workshop on Crafting didactic materials based on IMS LD: from Requirements to Evaluation at ICALT08*, Cantanbria, Spain.

[Pahl et al., 2007] Pahl, C., Holohan, E., McMullen, D., and Melia, M. (2007). *Learning objects: theory, praxis, issues, and trends*, chapter Ontology-based Learning Objects in Learning Content Management Systems. Informing Science.

[Pahl and Melia, 2006] Pahl, C. and Melia, M. (2006). Semantic Modelling of Learning Objects and Instruction. In *Proceeding of the First European Conference on Technology Enhanced Learning (EC-TEL 2006)*. Springer-Verlag LNCS.

[Paquette et al., 2006] Paquette, G., Leonard, M., Lundren-Cayrol, K., Mihaila, S., and Gareau, D. (2006). Learning design based on graphical knowledge-modelling. *Journal of Educational Technology and Society*, 9(1):97–112.

[Pepper and Moore, 2001] Pepper, S. and Moore, G. (2001). Xml topic maps 1.0. Specification, TopicMaps.org.

[Persico, 1996] Persico, D. (1996). Courseware validation: a case study. *Journal of Computer Assisted Learning*, 12(4):232–244.

[Popma, 2003] Popma, R. (2003). Jet tutorial part 1 (introduction to jet). http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html.

[Reigeluth, 1983a] Reigeluth, C. M., editor (1983a). *Instructional-Design: Theories and Models*. Lawrence Erlbaum Associates, Publishers, New Jersey, USA.

[Reigeluth, 1983b] Reigeluth, C. M. (1983b). *Instructional-Design: Theories and Models*, chapter Instructional Design: What is it and why is it?, pages 3–54. Lawrence Erlbaum Associates, Publishers.

[Reigeluth, 1999a] Reigeluth, C. M. (1999a). *Instructional Design: Theories and Models*, volume 2. Lawrence Erlbaum Associates, Publishers.

[Reigeluth, 1999b] Reigeluth, C. M., editor (1999b). *Instructional Design: Theories and Models*, volume 2, chapter The Elaboration Theory: Guidance for Scope and Sequencing Decisions, pages 425–453. Lawrence Erlbaum Associates, Publishers.

[RELOAD Project, 2005] RELOAD Project (2005). The RELOAD Metadata and Content Packaging Editor. Available from: http://www.reload.ac.uk/editor.html.

[Richters, 2001] Richters, M. (2001). *A precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universität Bremen, Fachbereich Mathematik und Informatik.

[Rosmalen et al., 2006] Rosmalen, P. V., Vogten, H., Es, R. V., Passier, H., Poelmans, P., and Koper, R. (2006). Authoring a full life cycle model in standards-based, adaptive e-learning. *Journal of Educational Technology and Society*, 9(1):72–83.

[Samples, 2002] Samples, J. W. (2002). The pedagogy of technology - our next frontier? *Connexions*, 14(2):4–5.

[Schmidt, 2006] Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31.

[Sicilia, 2005] Sicilia, M.-A. (2005). *Intelligent Learning Infrastructure for Knowledge Intensive Organizations: A Semantic Web Perspective*, chapter Ontology-Based Competency Management: Infrastructures for the Knowledge Intensive Learning Organization, pages 302–324. Idea Group.

[Sicilia, 2006] Sicilia, M.-A. (2006). Semantic learning designs: recording assumptions and guidelines. *British Journal of Educational Technology*, 37(3):331–350.

[Sicilia, 2007] Sicilia, M.-A. (2007). On the general structure of ontologies in instructional models. In *In proceedings of the fourth Simposio Pluridisciplinar sobre Diseño, Evaluación y Desarrollo de Contenidos Educativos Reutilizables (SPDECE)*, Bilbao, Spain.

[Simon et al., 2005] Simon, B., Massart, D., van Assche, F., Ternier, S., Duval, E., Brantner, S., Olmedilla, D., and Miklos, Z. (2005). A simple query interface for interoperable learning repositories. In *Proceedings of the 14th International World Wide Web Conference*, Chiba, Japan. ACM.

[Sommerville, 2004] Sommerville, I. (2004). *Software Engineering*. Addison Wesley, 7 edition.

[Sosteric and Hesemeier, 2002] Sosteric, M. and Hesemeier, S. (2002). When a learning object is not an object: A first step towards a theory of learning object. *International Review of Research in Open and Distance Learning Journal*, 3(2).

[Stash et al., 2004] Stash, N., Cristea, A., and DeBra, P. (2004). Authoring of learning styles in adaptive hypermedia: Problems and solutions. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 114–123, New York, NY, USA. ACM.

[Steele, 1990] Steele, G. (1990). *Common LISP*. Digital Press.

[Steinburg et al., 2008] Steinburg, D., Budinsky, F., Paternostro, M., and Merks, E. (2008). *Eclipse Modeling Framework*. Pearson Education, 2nd edition.

[Stollberg et al., 2006] Stollberg, M., Moran, M., Cabral, L., Norton, B., and Domingue, J. (2006). Experiences from semantic web services tutorials. In *Semantic Web Education and Training Workshop at ASWC2006*.

[Su et al., 2005] Su, J.-M., Tseng, S.-S., Weng, J.-F., Chen, K.-T., Liu, Y.-L., and Tsai, Y.-T. (2005). An Object Based Authoring Tool for Creating SCORM Compliant Course. In *19th International Conference on Advanced Information Networking and Applications*, volume 1, pages 209–214. IEEE.

[Szyperski, 2002] Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, second edition.

[Ternier et al., 2008] Ternier, S., Massart, D., Campi, A., Guinea, S., Ceri, S., and Duval, E. (2008). Interoperability for searching learning object repositories. *D-Lib Magazin*, 14(1/2).

[Ullrich, 2005] Ullrich, C. (2005). Course generation based on HTN planning. In *Proceeding of the Thirteenth Annual Workshop of the SIG Adaptivity and User Modeling in Interactive Systems*, pages 74–79.

[USE, 2008] USE, U. (2008). Use: A uml-based specification environment. http://www.db.informatik.uni-bremen.de/projects/USE/.

[Van der Vegt and Koper, 2005] Van der Vegt, W. and Koper, R. (2005). Copperauthor. http:hdl.handle.net1820492.

[VanAssche, 2007] VanAssche, F. (2007). Linking learning resources to curricula by using competencies. In *In Proceedings of the First International Workshop on Learning Object Discovery and Exchange*. CEUR Workshop Proceedings.

[Vassileva and Deters, 1998] Vassileva, J. and Deters, R. (1998). Dynamic courseware generation on the WWW. *British Journal of Educational Technology*, 29(1):5–14.

[W3C, 2004] W3C (2004). Owl web ontology language guide.

[Wade and Ashman, 2007] Wade, V. P. and Ashman, H. (2007). Evolving the infrastructure for technology-enhanced distance learning. *Internet Computing*, 11(3):16–18.

[Wahler et al., 2006] Wahler, M., Koehler, J., and Brucker, A. D. (2006). Model-driven constraint engineering. In Chiorean, D., Demuth, B., Giese, M., and Warmer, J. B., editors, *Proceedings of the Sixth OCL Workshop OCL for (Meta-)Models in Multiple Application Domains (OCLApps 2006)*, number 1863-2122. ECEASST.

[Warmer and Kleppe, 2006] Warmer, J. and Kleppe, A. (2006). Octopus. http://www.klasse.nl/octopus/index.html.

[Warmer and Kleppe, 2003] Warmer, J. B. and Kleppe, A. (2003). *The Object Constraint Language*. Addison Wesley, 2 edition.

[Weber and Brusilovsky, 2001] Weber, G. and Brusilovsky, P. (2001). Elm-art: An adaptive versatile system for web based instruction. *International Journal of Artificial Intelligence in Education*, 12:351–384.

[White, 2004] White, S. A. (2004). Introduction to BPMN. Technical report.

[Wiley, 2001] Wiley, D. A. (2001). *The Instructional use of Learning Objects*, chapter Connecting Learning Objects to Instructional Design Theory: A definition, a methaphor and a taxonomy. Association for Educational Communications and Technology.

[Yang et al., 2005] Yang, J.-T. D., Chen, W.-C., Tsai, C.-Y., and Chao, M.-S. (2005). An Ontological Model for SCORM-Compliant Authoring Tools. *Journal of Information Science and Engineering*, 21(5):891–909.

[Yang and Tsai, 2003] Yang, J.-T. D. and Tsai, C.-Y. (2003). An Implementation of compliant Learning Content Management System - Content Repository Management System. In *Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT03)*. IEEE Computer Society.

Appendix A

Case-Study Documents

```
<?xml version=”1.0”?>
<rdf:RDF
    xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
    xmlns:skos=”http://www.w3.org/2004/02/skos/core.rdf#”
    xmlns:owl=”http://www.w3.org/2002/07/owl#”
    xmlns:xsd=”http://www.w3.org/2001/XMLSchema#”
    xmlns=”http://www.computing.dcu.ie/~mmelia/ontologies/databases-skos.owl#”
    xmlns:rdfs=”http://www.w3.org/2000/01/rdf-schema#”
  xml:base=”http://www.computing.dcu.ie/~mmelia/ontologies/databases-skos.owl”>
  <owl:Ontology rdf:about=””>
    <owl:imports rdf:resource=””/>
  </owl:Ontology>
  <skos:Concept rdf:ID=”First_Normal_Form”>
    <skos:broader>
      <skos:Concept rdf:ID=”Normalisation”>
        <skos:narrower>
          <skos:Concept rdf:ID=”Fourth_Normal_Form”>
            <skos:broader rdf:resource=”#Normalisation”/>
          </skos:Concept>
        </skos:narrower>
        <skos:narrower rdf:resource=”#First_Normal_Form”/>
        <skos:subject>
          <skos:Concept rdf:ID=”Multivalued_Dependency”>
            <skos:isSubjectOf rdf:resource=”#Normalisation”/>
            <skos:related>
              <skos:Concept rdf:ID=”BCNF”>
                <skos:broader rdf:resource=”#Normalisation”/>
                <skos:related rdf:resource=”#Multivalued_Dependency”/>
              </skos:Concept>
            </skos:related>
          </skos:Concept>
        </skos:subject>
        <skos:narrower rdf:resource=”#BCNF”/>
        <skos:narrower>
          <skos:Concept rdf:ID=”Third_Normal_Form”>
            <skos:broader rdf:resource=”#Normalisation”/>
            <skos:related>
              <skos:Concept rdf:ID=”Functional_Dependency”>
                <skos:related>
                  <skos:Concept rdf:ID=”Second_Normal_Form”>
                    <skos:related rdf:resource=”#Functional_Dependency”/>
                    <skos:broader rdf:resource=”#Normalisation”/>
                  </skos:Concept>
                </skos:related>
                <skos:related rdf:resource=”#Third_Normal_Form”/>
                <skos:isSubjectOf rdf:resource=”#Normalisation”/>
              </skos:Concept>
            </skos:related>
          </skos:Concept>
        </skos:narrower>
        <skos:subject rdf:resource=”#Functional_Dependency”/>
        <skos:broader>
          <skos:Concept rdf:ID=”Database_Systems”>
            <skos:broader>
              <skos:Concept rdf:ID=”Information_Systems”>
                <skos:narrower rdf:resource=”#Database_Systems”/>
                <rdfs:comment rdf:datatype=”http://www.w3.org/2001/XMLSchema#string”
                ></rdfs:comment>
              </skos:Concept>
            </skos:broader>
            <skos:narrower>
              <skos:Concept rdf:ID=”Database_Architecture”>
                <skos:broader rdf:resource=”#Database_Systems”/>
                <skos:narrower>
                  <skos:Concept rdf:ID=”Three_Layer_Model”>
                    <skos:broader rdf:resource=”#Database_Architecture”/>
                  </skos:Concept>
                </skos:narrower>
              </skos:Concept>
            </skos:narrower>
            <skos:narrower>
              <skos:Concept rdf:ID=”System_Catalog”>
                <skos:related>
                  <skos:Concept rdf:ID=”DML”>
                    <skos:related rdf:resource=”#System_Catalog”/>
                    <skos:broader>
                      <skos:Concept rdf:ID=”SQL”>
                        <skos:broader rdf:resource=”#Database_Systems”/>
                        <skos:narrower rdf:resource=”#DML”/>
                        <skos:narrower>
                          <skos:Concept rdf:ID=”DDL”>
                            <skos:broader rdf:resource=”#SQL”/>
                          </skos:Concept>
                        </skos:narrower>
                      </skos:Concept>
                    </skos:broader>
                    <skos:narrower>
```

Figure A.1: Databases domain model defined using SKOS (part 1)

```
                      <skos:Concept rdf:ID="SQL_Select">
                        <skos:related>
                          <skos:Concept rdf:ID="Relational_Modelling">
                            <skos:broader rdf:resource="#Database_Systems"/>
                            <skos:narrower>
                              <skos:Concept rdf:ID="Relational_Calculus">
                                <skos:broader rdf:resource="#Relational_Modelling"/>
                              </skos:Concept>
                            </skos:narrower>
                            <skos:narrower>
                              <skos:Concept rdf:ID="Relational_Algebra">
                                <skos:broader rdf:resource="#Relational_Modelling"/>
                              </skos:Concept>
                            </skos:narrower>
                            <skos:related rdf:resource="#SQL_Select"/>
                          </skos:Concept>
                        </skos:related>
                        <skos:broader rdf:resource="#DML"/>
                      </skos:Concept>
                    </skos:narrower>
                    <skos:narrower>
                      <skos:Concept rdf:ID="SQL_Views">
                        <skos:broader rdf:resource="#DML"/>
                      </skos:Concept>
                    </skos:narrower>
                  </skos:Concept>
                </skos:related>
                <skos:broader rdf:resource="#Database_Systems"/>
              </skos:Concept>
            </skos:narrower>
            <skos:narrower rdf:resource="#SQL"/>
            <skos:narrower>
              <skos:Concept rdf:ID="Storage_Structures">
                <skos:narrower>
                  <skos:Concept rdf:ID="Indexing">
                    <skos:broader rdf:resource="#Storage_Structures"/>
                  </skos:Concept>
                </skos:narrower>
                <skos:narrower>
                  <skos:Concept rdf:ID="Hashing">
                    <skos:broader rdf:resource="#Storage_Structures"/>
                  </skos:Concept>
                </skos:narrower>
                <skos:broader rdf:resource="#Database_Systems"/>
              </skos:Concept>
            </skos:narrower>
            <skos:narrower rdf:resource="#Normalisation"/>
            <skos:narrower>
              <skos:Concept rdf:ID="JDBC">
                <skos:broader>
                  <skos:Concept rdf:ID="Java_Programming">
                    <skos:narrower rdf:resource="#JDBC"/>
                  </skos:Concept>
                </skos:broader>
                <skos:broader rdf:resource="#Database_Systems"/>
              </skos:Concept>
            </skos:narrower>
            <skos:narrower>
              <skos:Concept rdf:ID="ER_Modelling">
                <skos:broader>
                  <skos:Concept rdf:ID="Modelling">
                    <skos:narrower rdf:resource="#ER_Modelling"/>
                  </skos:Concept>
                </skos:broader>
                <skos:broader rdf:resource="#Database_Systems"/>
              </skos:Concept>
            </skos:narrower>
            <skos:narrower rdf:resource="#Relational_Modelling"/>
          </skos:Concept>
        </skos:broader>
        <skos:narrower>
          <skos:Concept rdf:ID="Fifth_Normal_Form">
            <skos:broader rdf:resource="#Normalisation"/>
          </skos:Concept>
        </skos:narrower>
        <skos:narrower rdf:resource="#Second_Normal_Form"/>
      </skos:Concept>
    </skos:broader>
  </skos:Concept>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.3, Build 418) http://protege.stanford.edu -->
```

Figure A.2: Databases domain model defined using SKOS

Appendix B

Implementation Documents

```
rule concept2concept{
    from
        pt: XML!Element(
            pt.name = 'skos:Concept'
        )
    to
        inf : MIKAEL!Concept(
            name <- pt.children->select(e|e.name='rdf:ID')->first().value
        )
}

rule SKOSRelationshipBroader2conceptRelationship{
    from
        ptRel1: XML!Element(
            ptRel1.name = 'skos:broader'
        )
    using{
           --merge all the input models
            b:Set(XML!Element) = XML!Element.allInstancesFrom('IN');
        }
    to
        infRelBro : MIKAEL!ConceptRelationship(
            label<-'narrowerThan',
            type <- #NARROWER,
            target<-thisModule.resolveTemp(ptRel1.parent,'inf'),
            source<- if ptRel1.children->select(e|e.name='skos:Concept').notEmpty() then
                    thisModule.resolveTemp(ptRel1.children->select(e|e.name='skos:Concept')->first(),'inf')
                else

                    thisModule.resolveTemp(
                        b->select(e|e.name = 'skos:Concept')
                            ->select(e|e.children->select(e2|e2.value=
                                    ptRel1.getPrevCreatedConceptName()).notEmpty())->first(), 'inf')
                endif
            )
}

 rule SKOSRelationshipRelated2conceptRelationship{
    from
        ptRelRel: XML!Element(
            ptRelRel.name = 'skos:related'
        )
    using{
           --merge all the input models
            b:Set(XML!Element) = XML!Element.allInstancesFrom('IN');
        }
    to
        infRelRel : MIKAEL!ConceptRelationship(
            label<-'RelatedTo',
            type <- #RELATED,
            source<-thisModule.resolveTemp(ptRelRel.parent,'inf'),
            target<- if ptRelRel.children->select(e|e.name='skos:Concept').notEmpty() then
                    thisModule.resolveTemp(ptRelRel.children->select(e|e.name='skos:Concept')->first(),'inf')
                else

                    thisModule.resolveTemp(
                        b->select(e|e.name = 'skos:Concept')
                        ->select(e|e.children->select(e2|e2.value=
                            ptRelRel.getPrevCreatedConceptName()).notEmpty())->first(), 'inf')
                endif
            )
}
```

Figure B.1: Excerpt from SKOS to CAVIAr domain model transformation defined in ATL

```
module CG2SC; -- Module Template
create OUT : SCORM from IN : COURSE;


helper def: getCourseWare(): COURSE!CoursewareModel =
    COURSE!CoursewareModel.allInstances()->first();


-- ============================================================================
-- RULES
-- ============================================================================

    rule xx{
        from
            courseWare: COURSE!CoursewareModel

        using{
            topics:Set(COURSE!Topic)=courseWare.topics;

        }

        to

            manifest: SCORM!Manifest(
                organizations <- organizations,
                resources <- courseWare.topics->collect(e|e.resources),
                metadata <- metadata
            ),

            metadata: SCORM!Metadata(
                    schema <- 'ADL SCORM',
                    schemaversion <- '1.1'
            ),

            organizations:SCORM!Organizations(
                title <- 'new mikael Organizations',
                organizations <- Set{organization}
            ),

            organization: SCORM!Organization(
                title <- 'new mikael Organization' ,
                items <- topics -> collect(e|thisModule.resolveTemp(e,'item'))
            )



}


    rule Topic2Item{
        from
            topic:COURSE!Topic

        using{
            topicRelations : Set(COURSE!TopicRelationship)=
                COURSE!CoursewareModel.allInstances()
                ->first().topicRelations
                ->select(e|e.type = #CONTAINS);
        }

        to
            item: SCORM!Item(
                title<-topic.name,
                items <-
                    topicRelations
                    -> select(e | e.source = topic )
                    ->collect(e| e.target),
                resources<-topic.resources
            )
    }

    rule LearningObject2Resources{
        from
            lo:COURSE!LearningObject
        to
            resource:SCORM!Resource(
                files<-file ,
                identifier<-lo.metadata.classifications->first().concept.name
            ),

            file:SCORM!File(
                location<-lo.metadata.technical.location
            )
    }
```

Figure B.2: Excerpt from CAVIAr courseware model to SCORM model defined in ATL

```
module SC2XML; -- Module Template
create OUT : XML  from IN : SCORM;

rule resource{
    from
        r:SCORM!Resource
    to
        element:XML!Element(
            name <-'resource',
            children<-id,
            children <-r.files,
            children <- href
            ),

        id : XML!Attribute (
            name <- 'identifier',
            value <- r.identifier
        ),

        href : XML!Attribute (
            name <- 'href',
            value <- r.files->first().location
        )


}

rule file{
    from
        f:SCORM!File
    to
        xmlf:XML!Element(
            name<-'file',
            children<-location),

        location : XML!Attribute (
            name <- 'href',
            value <- f.location
        )
}


rule sss{
    from
        manifest : SCORM!Manifest
    to
        root : XML!Root (
            name <- 'manifest',

            children <- xmlns,
            children <- version,
            children <- identifier,
            children <- resources,
            children <- manifest.metadata,
            children <- manifest.organizations
        ),

        resources: XML!Element(
            name<-'resources',
            children<-manifest.resources
        ),

        xmlns : XML!Attribute (
            name <- 'xmlns',
            value <- 'http://www.imsglobal.org/xsd/imscp_v1p1'
        ),

        version : XML!Attribute (
            name <- 'version',
            value <- '1.1'
        ),

        identifier: XML!Attribute(
            name <- 'identifier',
            value <- 'mikael scorm manifest'
        )
}
```

Figure B.3: SCORM model to SCORM manifest XML transformation definition in ATL (part 1)

```
rule organizations{
    from
        organizations: SCORM!Organizations
    to
        element:XML!Element(
            name <- 'organizations',
            children <- title,
            children <- organizations.organizations
        ),
        title: XML!Element(
            name <-'title',
            children <- thisModule.Text(organizations.title)
        )
}

rule organization{
    from
        organization:SCORM!Organization
    to
        element:XML!Element(
            name <- 'organization',

            children <- title,

            children<- organization.items
        ),
        title: XML!Element(
            name <-'title',
            children <- thisModule.Text(organization.title)
        )
}

rule item{
    from
        item:SCORM!Item
    to
        element:XML!Element(
            name<- 'item',
            children <- identifierref,
            children <- identifier,
            children <- title,
            children <- item.items

        ),
        identifierref: XML!Attribute(
            name <- 'identifierref',
            value <- item.title
        ),
        identifier: XML!Attribute(
            name <- 'identifier',
            value <- item.title
        ),
        title: XML!Element(
            name <-'title',
            children <- thisModule.Text(item.title)
        )
}
```

Figure B.4: SCORM model to SCORM manifest XML transformation definition in ATL (part 2)

```
rule metadata{
    from
        metadata:SCORM!Metadata
    to
        element:XML!Element(
            name<- 'metadata',
            children <- schema,
            children <- schemaversion
        ),
        schema: XML!Element(
            name <- 'schema',
            children <- thisModule.Text(metadata.schema)
        ),
        schemaversion: XML!Element(
            name <- 'schemaversion',
            children <- thisModule.Text(metadata.schemaversion)
        )
}



rule Text(txtValue:String){
    to
        txt: XML!Text(
            value<-txtValue
        )
    do{
        txt;
    }
}

rule Attribute(attrName : String, attrValue : String) {
    to
        attr : XML!Attribute (
            name <- attrName,
            value <- attrValue
        )
    do {
        attr;
    }
}
```

Figure B.5: SCORM model to SCORM manifest XML transformation definition in ATL (part 3)

```
module low2high; -- Module Template
create OUT : COURSE from IN : DOMAIN;

---------- helper ----------

-- find all goal concepts
helper def: getGoalConcepts(): Sequence(DOMAIN!Concept) =
    --let Competency!DOMAIN ->
    DOMAIN!Competency.allInstances()
            -> select(e| e.refImmediateComposite().oclIsTypeOf(DOMAIN!Goal))
            -> collect(e|e.concept)
            -> asSet();

helper def: getGoalLearningObjects(c:Domain!Concept): Set(DOMAIN!Resource) =
    DOMAIN!LearningObject.allInstances();
            --> select(e| c = e.metadata.classifications->first().concept);


-- find all inersection goal concepts, which are aggregationlevel1
helper def: getIntersectionConcepts(): Sequence(DOMAIN!Concept) =
    DOMAIN!Goal.allInstances()
        -> iterate (e ; res: Sequence(DOMAIN!Concept)= Sequence{} |
             if res.notEmpty() then
                 res -> asSet().intersection(e.hasKnowledge
                      -> collect(e_competency | e_competency.concept)->asSet())
             else
                 res.union(e.hasKnowledge -> collect(e_competency | e_competency.concept))
             endif
);

-- find all union goal concepts, which are aggregationlevel1
helper def: getUnionConcepts(): Sequence(DOMAIN!Concept) =
    DOMAIN!Goal.allInstances()
        -> iterate (e ; res: Sequence(DOMAIN!Concept)= Sequence{} |
             if res.notEmpty() then
                 res -> asSet().union(e.hasKnowledge -> collect(e_competency | e_competency.concept)->asSet())
             else
                 res.union(e.hasKnowledge -> collect(e_competency | e_competency.concept))
             endif
);

-- find all concepts with a relation to the intersection concepts
helper def: getSubConceptsOfGoalUnion(): Sequence(DOMAIN!Concept) =
    thisModule.getUnionConcepts()
        -> iterate (e ; res: Sequence(DOMAIN!Concept)= Sequence{} |
             if res.notEmpty() then
                 res -> asSet().union(e.relations->select(k|k.type<>#RELATED or  k.type<>#PREREQUISITE)
                     -> collect(relation| relation.target)->asSet())
             else
                 res.union(e.relations->select(k|k.type<>#RELATED or  k.type<>#PREREQUISITE)
                      -> collect(relation | relation.target))
             endif
);

-- find all sub concepts of aggregationlevel1 concepts. which are aggregationlevel2
helper def: getAggregationLevel2Concepts(): Sequence(DOMAIN!Concept) =
    thisModule.getIntersectionConcepts()
        -> iterate (e; res: Sequence(DOMAIN!ConceptRelationship)=Sequence{} |
                 res.union(e.relations)
             )
        -> select(e_conceptRelationship | e_conceptRelationship.type=#NARROWER or
              e_conceptRelationship.type=#PREREQUISITE)
        -> collect (e|e.target)-> asSet();

--- get the concept by name
helper def: getNewConceptByName(p: String):COURSE!Concept =
    COURSE!Concept.allInstances() -> select(e| e.name = p).first();

helper def: getAllSubtopics(): Sequence(DOMAIN!Concept) =
    thisModule.getGoalConcepts()
        -> iterate (e; res: Sequence(DOMAIN!ConceptRelationship)=Sequence{} |
                 res.union(e.relations)
             )
        -> select(e_conceptRelationship | e_conceptRelationship.type=#PREREQUISITE)
        -> collect(e_conceptRelationship1 |e_conceptRelationship1.target);

----------- rule ------------
```

Figure B.6: ATL transformation definition, transforming the learning context model into a courseware model (part 1)

```
rule caviarCreate{
    from
        caviar:DOMAIN!Caviar
    to
        newCaviar:COURSE!Caviar(
            coursewareModel <- caviar.learningContext,
            learningResourceModel <- caviar.learningResourceModel
        )

}



rule coursewareCreate{
    from
        lc:DOMAIN!LearningContext
    to

        c:COURSE!CoursewareModel(
            topics<-lc.domainModel.concepts,
            topicRelations<-lc.domainModel.conceptRelations
        )
}


rule IntersectionGoalConcept2topic{
    from
        concept:DOMAIN!Concept(
            thisModule.getUnionConcepts().includes(concept)
        )
    to

        newTopic:COURSE!Topic(
            name <- concept.name,
            aggregationLevel <-1,
            resources <-thisModule.getGoalLearningObjects(concept)
        )


}

rule IntersectionGoalConcept2subtopic{
    from
        subConcept:DOMAIN!Concept(
            thisModule.getSubConceptsOfGoalUnion().includes(subConcept)
            )
    to

        newSubTopic:COURSE!Topic(
            name <- subConcept.name,
            aggregationLevel <-2,
            resources <-thisModule.getGoalLearningObjects(subConcept)
        ),

        subTopicRelation:COURSE!TopicRelationship(
            label<-subConcept.incomingRelations ->select(e|e.type<>#RELATED or
e.type<>#PREREQUISITE)->first().label
           --label<-'containedTopic'

        )

}
rule conceptRelationships2TopicRelationships{
    from
        subConceptRelation:DOMAIN!ConceptRelationship(
            thisModule.getSubConceptsOfGoalUnion()
                ->iterate (e; res: Sequence(DOMAIN!ConceptRelationship)=Sequence{} |
                res.union(e.incomingRelations->select(
                    k| --(e.type<>#RELATED or e.type<>#PREREQUISITE) and
DOES NOT WORK BUT MIGHT CAUSE PROBLEMS WITHOUT
                    thisModule.getUnionConcepts()->includes(k.source)
                    ))) ->includes(subConceptRelation)
            )
    to

        subTopicRelation:COURSE!TopicRelationship(
            label<-'containedTopic',
            type<-#CONTAINS,
            target<-subConceptRelation.target,
            source<-subConceptRelation.source

        )
}
```

Figure B.7: ATL transformation definition, transforming the learning context model into a courseware model (part 2)

```
rule learningResourceModel2LearningResourceModel{
    from
        lrm:DOMAIN!LearningResourceModel
    to
        nlrm:COURSE!LearningResourceModel(
            resources <- lrm.resources
        )
}

rule LO2LO{
    from
        lo:DOMAIN!LearningObject
    to
        nlo:COURSE!LearningObject(
            metadata <- lo.metadata
        )
}

rule meta2meta{
    from
        m:DOMAIN!Metadata
    to
        nm:COURSE!Metadata(
            general<-m.general,
            educational<-m.educational,
            technical<-m.technical,
            classifications<-m.classifications

        )
}



rule gen2gen{
    from
        gen:DOMAIN!General
    to
        ngen:COURSE!General(
            title <- gen.title,
            coverage <- gen.coverage,
            structure <- gen.structure,
            description <- gen.description,
            aggregationLevel <- gen.aggregationLevel

        )
}

rule edu2edu{
    from
        edu:DOMAIN!Educational
    to
        nedu:COURSE!Educational(
            interactionType <- edu.interactionType,
            interactivityLevel <- edu.interactivityLevel,
            semanticDensity <- edu.semanticDensity
        )
}

rule tech2tech{
    from
        tech:DOMAIN!Technical
    to
        ntech:COURSE!Technical(
            durcation<- tech.durcation,
            location <- tech.location,
            format <- tech.format
        )
}
```

Figure B.8: ATL transformation definition, transforming the learning context model into a courseware model (part 3)

```
module lowhigh2caviar; -- Module Template
create OUT : CAVIAR from IN : CAVIAR, IN2: CAVIAR;

---------- helper ----------

-- find all goal concepts
helper def: getGoalConcepts(): Sequence(DOMAIN!Concept) =
    --let Competency!DOMAIN ->
    CAVIAR!Competency.allInstances()
            -> select(e| e.refImmediateComposite().oclIsTypeOf(DOMAIN!Goal))
            -> collect(e|e.concept)
            -> asSet();

-- find all inersection goal concepts, which are aggregationlevel1
helper def: getIntersectionConcepts(): Sequence(DOMAIN!Concept) =
    CAVIAR!Goal.allInstances()
        -> iterate (e ; res: Sequence(CAVIAR!Concept)= Sequence{} |
            if res.notEmpty() then
                res -> asSet().intersection(e.hasKnowledge
                    -> collect(e_competency | e_competency.concept)->asSet())
            else
                res.union(e.hasKnowledge -> collect(e_competency | e_competency.concept))
            endif
);


-- find all sub concepts of aggregationlevel1 concepts. which are aggregationlevel2
helper def: getAggregationLevel2Concepts(): Sequence(CAVIAR!Concept) =
    thisModule.getIntersectionConcepts()
        -> iterate (e; res: Sequence(CAVIAR!ConceptRelationship)=Sequence{} |
                res.union(e.relations)
            )
        -> select(e_conceptRelationship | e_conceptRelationship.type=#NARROWER or
             e_conceptRelationship.type=#PREREQUISITE)
        -> collect (e|e.target)
        -> asSet();

--- get the concept by name
helper def: getNewConceptByName(p: String):COURSE!Concept =
    COURSE!Concept.allInstances() -> select(e| e.name = p).first();

helper def: getAllSubtopics(): Sequence(CAVIAR!Concept) =
    thisModule.getGoalConcepts()
        -> iterate (e; res: Sequence(CAVIAR!ConceptRelationship)=Sequence{} |
                res.union(e.relations)
            )
        -> select(e_conceptRelationship | e_conceptRelationship.type=#PREREQUISITE)
        -> collect(e_conceptRelationship1 |e_conceptRelationship1.target);


-------------CAVIAR Transformation-----------------------

rule caviar2caviar{
    from
        caviarModel: CAVIAR!Caviar(
            caviarModel.learningContext.oclIsUndefined()
        )
    using{
            a:Set(CAVIAR!Caviar) = CAVIAR!CoursewareModel.allInstancesFrom('IN2')
                ->union(CAVIAR!LearningContext.allInstancesFrom('IN'))
                ->union(CAVIAR!LearningResourceModel.allInstancesFrom('IN'));
        }
    to
        ncaviar: CAVIAR!Caviar(
            learningContext <- a->select(e|e.oclIsTypeOf(CAVIAR!LearningContext))->first(),
            coursewareModel <- a->select(k|k.oclIsTypeOf(CAVIAR!CoursewareModel))->first(),
            learningResourceModel <- a->select(j|j.oclIsTypeOf(CAVIAR!LearningResourceModel))->first()


        )
}


----------LEARNING CONTEXT TRANSFORMATION----------------------

rule lc2lc{
    from
        lc: CAVIAR!LearningContext
    to
        nlc: CAVIAR!LearningContext(
            domainModel <- lc.domainModel,
            learnerStereotypes <- lc.learnerStereotypes
        )
}
```

Figure B.9: ATL transformation definition, merging the learning context model with courseware model (part 1)

```
rule dm2dm{
     from
          dm: CAVIAR!DomainModel
     to
          ndm: CAVIAR!DomainModel(
               concepts<-dm.concepts,
               conceptRelations <- dm.conceptRelations
          )
}


rule learnerStereotype2learnerStereotype{
     from
          learnerStereotype: CAVIAR!LearnerStereotype
     to
          newLearnerStereotype: CAVIAR!LearnerStereotype(
               knowledgeConstraints <- learnerStereotype.knowledgeConstraints,
               goalRelations <- learnerStereotype.goalRelations
          )
}

rule goal2goal{
     from
          goal:CAVIAR!Goal
     to
          newGoal:CAVIAR!Goal(
               hasKnowledge <- goal.hasKnowledge
               )
}

rule goalRelation2goalRelation{
     from
          goalR:CAVIAR!GoalRelationship
     to
          nGoalR:CAVIAR!GoalRelationship(
               label <- goalR.label,
               type <- goalR.type,
               source <- goalR.source,
               target <- goalR.target
          )
}

rule presumedKnowledge2presumedKnowledge{
     from
          pk:CAVIAR!PresumedKnowledge
     to
          npk:CAVIAR!PresumedKnowledge(
               hasKnowledge <- pk.hasKnowledge
          )
}



rule competency2competency{
     from
          compet:CAVIAR!Competency
     to
          nCompet:CAVIAR!Competency(
               level <- compet.level,
               concept <- compet.concept
          )
}

rule concept2concept{
     from
          c:CAVIAR!Concept
     to
          nc:CAVIAR!Concept(
               name <- c.name,
               synonyms <- c.synonyms
          )
}
```

Figure B.10: ATL transformation definition, merging the learning context model with courseware model (part 2)

```
rule ConceptRelationship2ConceptRelationship{
    from
        cr:CAVIAR!ConceptRelationship
    to
        ncr:CAVIAR!ConceptRelationship(
            label <- cr.label,
            type<- cr.type,
            source <- cr.source,
            target <- cr.target
        )
}

rule Synonym2Synonym{
    from
        sy:CAVIAR!Synonym
    to
        nsy:CAVIAR!Synonym(
            value <- sy.value
        )
}
----------COURSEWARE MODEL TRANSFORMATION------------
rule coursewaremodel2coursewaremodel{
    from
        cwm:CAVIAR!CoursewareModel
    to
        ncwm:CAVIAR!CoursewareModel(
            name <- cwm.name,
            topics <- cwm.topics,
            topicRelations <- cwm.topicRelations->asSet()
        )

}
rule topic2topic{
    from
        t:CAVIAR!Topic
    to
        nt:CAVIAR!Topic(
            name <- t.name,
            complete <- t.complete,
            aggregationLevel <- t.aggregationLevel,
            entryLearners <- t.entryLearners,
            resources <- t.resources
        )
}

rule topicRelationship2topicRelationship{
    from
        tr:CAVIAR!TopicRelationship
    to
        ntr:CAVIAR!TopicRelationship(
            label <- tr.label,
            type <- tr.type,
            source <- tr.source,
            target <- tr.target
        )
}
rule entryLearner2entryLearner{
    from
        el:CAVIAR!EntryLearner
    to
        nel:CAVIAR!EntryLearner(
            learnerStereotype <- el.learnerStereoType,
            greaterThan <- el.greaterThan,
            lessThan <- el.lessThan
        )
}
---------------LEARNING RESOURCE MODEL TRANSFORMATION--------------
rule learningResourceModel2LearningResourceModel{
    from
        lrm:CAVIAR!LearningResourceModel
    to
        nlrm:CAVIAR!LearningResourceModel(resources <- lrm.resources)
}
rule LearningService2LearningService{
    from
        s:CAVIAR!LearningService
    to
        ns:CAVIAR!LearningService(
        )
}
```

Figure B.11: ATL transformation definition, merging the learning context model with courseware model (part 3)

```
rule LO2LO{
     from
          lo:CAVIAR!LearningObject
     to
          nlo:CAVIAR!LearningObject(
               metadata <- lo.metadata
          )
}
rule meta2meta{
     from
          m:CAVIAR!Metadata
     to
          nm:CAVIAR!Metadata(
               general<-m.general,
               educational<-m.educational,
               technical<-m.technical,
               classifications<-m.classifications

          )
}

rule relation2relation{
     from
          rel:CAVIAR!ResourceRelationship
     to
          nrel:CAVIAR!ResourceRelationship(
               kind <- rel.kind,
               label <- rel.label,
               source <- rel.source,
               target <- rel.target
          )
}

rule classif2classif{
     from
          class:CAVIAR!Classification
     to
          nclass:CAVIAR!Classification(
               label <- class.label,
               purpose <- class.purpose,
               concept <- class.concept,
               contentType <- class.contentType
          )
}


rule gen2gen{
     from
          gen:CAVIAR!General
     to
          ngen:CAVIAR!General(
               title <- gen.title,
               coverage <- gen.coverage,
               structure <- gen.structure,
               description <- gen.description,
               aggregationLevel <- gen.aggregationLevel

          )
}

rule edu2edu{
     from
          edu:CAVIAR!Educational
     to
          nedu:CAVIAR!Educational(
               interactionType <- edu.interactionType,
               interactivityLevel <- edu.interactivityLevel,
               semanticDensity <- edu.semanticDensity
          )
}

rule tech2tech{
     from
          tech:CAVIAR!Technical
     to
          ntech:CAVIAR!Technical(
               durcation<- tech.durcation,
               location <- tech.location,
               format <- tech.format
          )
}
```

Figure B.12: ATL transformation definition, merging the learning context model with courseware model (part 4)

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>

<plugin>

     <extension point="org.eclipse.ui.editorActions">
      <editorContribution
          id="org.eclipse.emf.validation.examples.general.editorContribution"
          targetID="Mikael2High.diagram.part.MIKAEL2DiagramEditorID">
         <menu
                label="CAVIAr Validation"
                path="additions"
                id="Course.Validation">
            <separator name="additions"/>
         </menu>

         <action
                label="%ValidateElementsAction.label"
                icon="$nl$/icons/elcl16/validate_co.gif"
                class="org.eclipse.emf.validation.examples.general.actions.BatchValidationDelegate"
                menubarPath="Course.Validation/additions"
                id="MIKAEL2hClientContext.ui.validateAction"/>
         <action
                label="%EnableLiveValidationAction.label"
                class="org.eclipse.emf.validation.examples.general.actions.EnableLiveValidationDelegate"
                menubarPath="Course.Validation/additions"
                id="MIKAEL2hClientContext.ui.enableLiveValidationAction"/>
        <action
               label="%ShowValidationEventsAction.label"
               class="org.eclipse.emf.validation.examples.general.actions.ShowValidationEventsDelegate"
               style="toggle"
               state="false"
               menubarPath="Course.Validation/additions"
               id="MIKAEL2hClientContext.ui.showValidationEventsAction"/>
      </editorContribution>
   </extension>

   <extension
         point="org.eclipse.ui.popupMenus">
      <viewerContribution
             targetID="Mikael2High.diagram.part.MIKAEL2hDiagramEditorID"
             id="org.eclipse.emf.validation.examples.general.viewerContribution">
         <menu
                label="%_UI_ValidationMenu_label"
                path="additions"
                id="org.eclipse.emf.validationMenuID">
            <separator name="additions"/>
         </menu>
         <action
                label="%ValidateElementsAction.label"
                icon="$nl$/icons/elcl16/validate_co.gif"
                class="org.eclipse.emf.validation.examples.general.actions.BatchValidationDelegate"
                menubarPath="org.eclipse.emf.validationMenuID/additions"
                id="MIKAEL2ClientContext.ui.validateAction"/>
         <action
                label="%EnableLiveValidationAction.label"
                class="org.eclipse.emf.validation.examples.general.actions.EnableLiveValidationDelegate"
                menubarPath="org.eclipse.emf.validationMenuID/additions"
                id="MIKAEL2ClientContext.ui.enableLiveValidationAction"/>
        <action
               label="%ShowValidationEventsAction.label"
               class="org.eclipse.emf.validation.examples.general.actions.ShowValidationEventsDelegate"
               style="toggle"
               state="false"
               menubarPath="org.eclipse.emf.validationMenuID/additions"
               id="MIKAEL2ClientContext.ui.showValidationEventsAction"/>
      </viewerContribution>
   </extension>




      <extension
          point="org.eclipse.emf.validation.constraintProviders"
          id="oclProvider">
      <!-- Custom constraint provider using OCL documents -->
      <constraintProvider
             class="org.eclipse.emf.validation.examples.ocl.OCLConstriantProvider2"
             category="mikael.validation"
             cache="false">

         <package namespaceUri="http://mikael.computing.dcu.ie/#2_0"/>

         <ocl path="constraints/instructionalDesign1.ocl"/>
      </constraintProvider>
  </extension>
```

Figure B.13: Plugin.xml file defining plugin to provide functionality for and handle OCL validation model (part 1)

```
   <extension
         point="org.eclipse.emf.validation.constraintBindings">
      <clientContext
            default="false"
            id="MIKAEL2ClientContext">
         <selector class=
            "org.eclipse.emf.validation.examples.general.constraints.ValidationDelegateClientSelector"/>
      </clientContext>
      <binding
            context="MIKAEL2ClientContext"
            category="mikael.validation"/>
   </extension>
   <extension
         point="org.eclipse.emf.validation.validationListeners">
      <listener class="org.eclipse.emf.validation.examples.general.listeners.ValidationListener">
         <clientContext id="MIKAEL2ClientContext"/>
      </listener>
   </extension>
   <extension
         point="org.eclipse.emf.validation.ui.UIRegisteredClientContext">
      <clientContext id="MIKAEL2ClientContext"/>
   </extension>
</plugin>
```

Figure B.14: Plugin.xml file defining plugin to provide functionality for and handle OCL validation model (part 2)

Appendix C

User Trial Survey

1. Participant Information

1. How many courses have you managed or delivered? - 0/1-3/4-10/10+
2. How many courses have you developed or adapted? - 0/1-3/4-10/10+
3. Please rate your knowledge on the following:
    - e-learning - no knowledge/familar/expert
    - e-learning authoring - no knowledge/familar/expert
    - personalised e-learning - no knowledge/familiar/expert

2. Courseware Construction Authoring Methodology

1. I understood the main CAVIAr courseware construction steps- Strongly disagree/disagree/neutral/agree/strongly agree
2. I understood the purpose of the domain model - Strongly disagree/disagree/neutral/agree/strongly agree
3. I understood the purpose of the learning context model - Strongly disagree/disagree/neutral/agree/strongly agree

4. I understood the purpose of the courseware model - Strongly disagree/disagree/neutral/agree/strongly agree
5. CAVIAr based courseware construction will offer cost and time savings - Strongly disagree/disagree/neutral/agree/strongly agree

3. Domain Model Editing

1. Creating a new MIKAEL project appeared simple - Strongly disagree/disagree/neutral/agree/strongly agree
2. Importing an external knowledge source appeared simple - Strongly disagree/disagree/neutral/agree/strongly agree
3. The domain model view was intuitive - Strongly disagree/disagree/neutral/agree/strongly agree
4. It appeared easy to use the domain model perspective - Strongly disagree/disagree/neutral/agree/strongly agree
5. Modelling notation used made sense - Strongly disagree/disagree/neutral/agree/strongly agree
6. It made sense to associate learning resources with domain concepts - Strongly disagree/disagree/neutral/agree/strongly agree
7. It appeared easy to add learning resources to domain concepts - Strongly disagree/disagree/neutral/agree/strongly agree
8. The metadata on LOs was sufficient to make a decision to add the LO or not - Strongly disagree/disagree/neutral/agree/strongly agree
9. Adding conceptual sequencing constraints appeared simple - Strongly disagree/disagree/neutral/agree/strongly agree

4. Learning Context Model Editing

1. Defining learner stereotypes for my course seemed simple - Strongly disagree/disagree/neutral/agree/strongly agree
2. It appeared easy to use the learning context perspective - Strongly disagree/disagree/neutral/agree/strongly agree
3. Terminology used was intuitive - Strongly disagree/disagree/neutral/agree/strongly agree
4. Terminology used was consistent - Strongly disagree/disagree/neutral/agree/strongly agree

5. Courseware Model Editing

1. I understand how the courseware model is created - Strongly disagree/disagree/neutral/agree/strongly agree
2. Creation of the courseware model is flexible - Strongly disagree/disagree/neutral/agree/strongly agree
3. A range of instructional designs can be used to create courses in MIKAEL - Strongly disagree/disagree/neutral/agree/strongly agree
4. The instructional design used in courseware can easily be changed - Strongly disagree/disagree/neutral/agree/strongly agree
5. I understand how to add learning resources to courseware topics - Strongly disagree/disagree/neutral/agree/strongly agree
6. Modelling notation made sense - Strongly disagree/disagree/neutral/agree/strongly agree
7. I understand how the courseware model adapts to learners - Strongly disagree/disagree/neutral/agree/strongly agree
8. I understand how to define sequencing restrictions on topics in my courseware - Strongly disagree/disagree/neutral/agree/strongly agree

9. I would be useful to be able to automatically check the coursware for problems - Strongly disagree/disagree/neutral/agree/strongly agree

10. It would be useful to check that instructional design has been applied correctly my courseware - Strongly disagree/disagree/neutral/agree/strongly agree

11. I could edit the courseware model in a way that would be inconsistent with the original course requirements - Strongly disagree/disagree/neutral/agree/strongly agree

6. Courseware Validation

1. The problems brought to my attention were actual problems - Strongly disagree/disagree/neutral/agree/strongly agree

2. I understand how validation works - Strongly disagree/disagree/neutral/agree/strongly agree

3. I understand how to define a validation model - Strongly disagree/disagree/neutral/agree/strongly agree

4. I would be more confident in the courseware I created after validating it - Strongly disagree/disagree/neutral/agree/strongly agree

5. I was waiting a long time to get my results - Strongly disagree/disagree/neutral/agree/strongly agree

7. Concluding Questions

1. I think the MIKAEL tool is useful - Strongly disagree/disagree/neutral/agree/strongly agree

2. I found creating a course confusing - Strongly disagree/disagree/neutral/agree/strongly agree

3. I think MIKAEL offers cost savings - Strongly disagree/disagree/neutral/agree/strongly agree

4. I think it will be easier to reuse learning resources with MIKAEL - Strongly disagree/disagree/neutral/agree/strongly agree
5. I think MIKAEL will increase the quaility of courses - Strongly disagree/disagree/neutral/agree/strongly agree

Appendix D

Acronyms

Term	*Explanation*
ACCT	Adaptive Courseware Construction Toolkit
ADL	Advanced Distributed Learning
AEH	Adaptive Educational Hypermedia
ALMA	Architecture Level Modiability Analysis
AM3	ATLAS MegaModel Management
AST	Abstract Syntax Tree
ATL	ATLAS Transformation Language
BPMN	Business Process Management Notation
CAIS	(DCU Course - Computer Applications Information Systems)
SCORM CAM	SCORM Content Aggregation Model
CASE	(DCU Course - Computer Applications Software Engineering)
CAVIAr	Courseware Authoring Validation Information Architecture
CBSD	Component Based Software Development
CDT	(Eclipse environment for C/C++ development)
CIM	Computationally Independent Model
CoCoA	Concept-based Course Analysis
IMS CP	IMS Content Packaging specification
DSL	Domain Specific Language
DSML	Domain Specific Modelling Language
DTD	Document Type Definition
ECore	(an EMF compatible language)
EMF	Eclipse Modelling Language
EML	Educational Modelling Language
GMF	Graphical Modelling Language
HTN	Hierarchical Task Analysis
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineers
IMS	Instructional Management Systems
IMS LD	IMS Learning Design Specification
ITS	Intelligent Tutoring System
JET	Java Emitter Template
LCMS	Learning Content Management System
LMS	Learning Management System
LO	Learning Object
IEEE LOM	IEEE Learning Object Metadata standard
M2M	(Eclipse model to model project)
M2T	(Eclipse model to text project)
MDA	Model Driven Architecture
MD(S)D	Model Driven (Software) Development

Term	*Explanation*
MDE	Model Driven Engineering
MIKAEL	Management Infrastructure for Knowledge-based Adaptive E-Learning
MOF	Meta-Object Facility
MOT	My Online Teacher
MTL	Model Transformation Language
NFQ	National Framework for Qualifications
OCL	Object Constraint Language
ODM	Ontology Definition Metamodel
OMG	Object Modelling Group
OWL	Web Ontology Language
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query/View/Transformation
RDF	Resource Description Framework
SCORM	Shareable Content Organisation Resource Model
SKOS	Simple Knowledge Organisational Structure
SCORM SN	SCORM Sequencing and Navigation
SQL	Structured Query Language
IMS SS	IMS Simple Sequencing specification
SWRL	Semantic Web Rule Language
TEL	Technology Enhanced Learning
UML	Unified Modelling Language
VLE	Virtual Learning Environment
W3C	World Wide Web Consortium
XML	Extensible Modelling Language