University of Crete

Department of Computer Science

"Towards standards based e-Learning tools and technologies"

By

George Tesseris

Master's Thesis

Heraklion, March 2010

University of Crete

Computer Science Department

Towards standards based e-Learning tools and technologies

By
George Tesseris

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science

Author: _____
George Tesseris, Department of Computer Science

Board of enquiry:

Supervisor: _____
Christos Nikolaou, Professor

Member: _____
Dimitrios Plexousakis, Professor

Member: _____
Michalis Kalogiannakis, Lecturer

Approved by: _____
Panos Trahanias, Professor,
Chairman of the Graduate Studies Committee

Heraklion, March 2010

"Towards standards based e-Learning tools and technologies"

George Tesseris

Master's Thesis

University of Crete
Computer Science Department

Abstract

The transition of today's global economy to one based on knowledge and information in combination with the recent advances in the field of information and communications technology (ICT), resulted in the explosive growth of e-Learning. Over the years, a significant number of e-Learning systems and learning resources were developed bringing, as customary in such cases, problems of interoperability. Many organizations (e.g. IMS, IEEE, AICC) began drafting a variety of different standards and specifications dealing, however, only with parts of the interoperability problem such as content packaging, content to LMS communication and metadata descriptions. The first effort towards addressing the full spectrum of the interoperability problem was Sharable Content Object Reference Model (SCORM), by the ADL. SCORM was widely accepted and endorsed by the e-Learning community and managed to achieve great success with version 1.2. The following version, however, failed to continue that trend. SCORM 2004, with the addition of learning content sequencing capabilities increased in size and complexity and the adoption rate was negatively impacted. In this thesis we present the design and implementation of an architecture for integrating the latest version of SCORM (SCORM 2004 4th Edition) into a pre-existent learning management system. The proposed architecture deals with the importing of any SCORM 2004 conformant content package, the runtime communication between content objects and the LMS and the sequencing of learning activities according to the rules defined in the content package. The implementation was successfully validated using the latest test suite provided by ADL.

"Προς εργαλεία και τεχνολογίες ηλεκτρονικής μάθησης βασισμένες σε πρότυπα"

Γεώργιος Τεσσέρης

Μεταπτυχιακή Εργασία

Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

Περίληψη

Η μετάβαση της σημερινής παγκόσμιας οικονομίας σε μια οικονομία βασισμένη στη γνώση και την πληροφορία, σε συνδυασμό με τις πρόσφατες προόδους στον τομέα των Τεχνολογιών Πληροφορικής και Επικοινωνιών (*ΤΠΕ*), είχε ως αποτέλεσμα την εκρηκτική ανάπτυξη της ηλεκτρονικής μάθησης. Με την πάροδο των ετών, ένας σημαντικός αριθμός συστημάτων ηλεκτρονικής μάθησης και εκπαιδευτικών πόρων αναπτύχθηκε, φέρνοντας, όπως συνηθίζεται σε αυτές τις περιπτώσεις, προβλήματα διαλειτουργικότητας. Πολλοί οργανισμοί (π.χ. IMS, IEEE, AICC) ξεκίνησαν τη σχεδίαση προτύπων και προδιαγραφών, που αφορούσαν ωστόσο μόνο σε μέρη του προβλήματος της διαλειτουργικότητας. Η πρώτη προσπάθεια για την αντιμετώπιση του προβλήματος συνολικά ήταν το Sharable Content Object Reference Model (SCORM), από την ADL. Το SCORM έγινε ευρέως αποδεκτό σημειώνοντας μεγάλη επιτυχία με την έκδοση 1.2. Η ακόλουθη έκδοση, ωστόσο, απέτυχε να γνωρίσει την ίδια επιτυχία. Το SCORM 2004, με την προσθήκη της δυνατότητας καθορισμού της *αλληλουχίας* παρουσίασης του εκπαιδευτικού υλικού αυξήθηκε σημαντικά σε μέγεθος και πολυπλοκότητα με αποτέλεσμα η διάδοση του να επηρεαστεί αρνητικά. Σε αυτήν τη διατριβή παρουσιάζουμε το σχεδιασμό και την υλοποίηση μιας αρχιτεκτονικής για την ενσωμάτωση της τελευταίας έκδοσης του SCORM (SCORM 2004 4η έκδοση) σε ένα προϋπάρχον σύστημα διαχείρισης μάθησης. Η προτεινόμενη αρχιτεκτονική χειρίζεται την εισαγωγή οποιουδήποτε πακέτο περιεχομένου που συμμορφώνεται με το SCORM 2004, την επικοινωνία μεταξύ του περιεχομένου και της πλατφόρμας διαχείρισης μάθησης και την αλληλουχία παρουσίασης του εκπαιδευτικού υλικού, σύμφωνα με τους κανόνες που ορίζονται στο πακέτο περιεχομένου. Η υλοποίηση της παραπάνω αρχιτεκτονικής πιστοποιήθηκε επιτυχώς χρησιμοποιώντας την αντίστοιχη πλατφόρμα πιστοποίησης που προσφέρει η ADL.

I would like to dedicate this thesis to my parents,

Ioannis and Stamatina,

and my brother, Anastasios

Acknowledgments

First and foremost, I would like to thank my thesis supervisor, Professor Christos Nikolaou, for his valuable guidance and encouragement during this work, and for the creative collaboration we had all these years.

I wish to express my sincere thanks to Professor Dimitris Plexousakis and Michalis Kalogiannakis for serving on my thesis committee, as well as for their constructive feedback and comments.

My best regards to the following, past and current, members of the Transformation Services Laboratory (TSL) for creating an inspiring environment to work in: Nikos Aggelidakis, Michalis Dimitriou, Mariana Karmazi, Efthimis Kartsonakis, Georgios Koutras, Pantelis Petridis, Alina Psycharaki, Georgios Stratakis and Kostas Tsikrikas.

I would also like to thank the eFront team (Michalis Makrigiannakis, Athanasios Papaggelis, Eleftheria Papatheodorou and Periklis Venakis) for their constant support and valuable suggestions.

Thanks to my boss, Dimitris Tsigos, for providing a positive and understanding working environment.

Thanks to all my friends for their patience, support and love. Special thanks to George Baryannis and Christina Lionoudaki.

My deepest gratitude goes to my family, to whom this thesis is dedicated to, for their unflagging love and motivational support.

Table of Contents

List of Figures

List of Tables

List of Listings

Chapter 1: **Introduction**

The transition of today's global economy to one based on knowledge and information brings new challenges to higher education and corporate training. Traditional educational settings do not satisfy the needs of the new lifelong learning (LLL) paradigm which demands continuous learning and skill development by the individual in order to stay competitive in the global market. Learning is shifting from instructor-centered and content-driven to learner-centered and learning process-driven. These challenges, in combination with the advances made in the field of information and communications technology (ICT), resulted in the explosive grow of e-Learning.

E-learning (Electronic learning) is a general term encompassing all training that is delivered with the assistance of a computer. Nowadays, an increasing number of academic institutions and companies provide e-Learning services through web-based learning management systems (LMSs). A typical learning management system provides features to distribute learning content, track learner progress and report learner performance. In addition, learning content is usually created using specialized content authoring tools and is stored in remote repositories that facilitate discovery and retrieval.

Over the years, the increasing interest in e-Learning led to the development of multiple LMSs, a considerable number of different repositories and several authoring tools from many vendors. These efforts encompassed differentiating technologies and architectures, which eventually posed great obstacles in interoperability and hindered any further advancement in the e-Learning domain. It became apparent that a set of standards needed to be defined, in order to offer a coherent and consistent basis for all efforts related to e-Learning. Prior to 1999, many organizations were drafting a variety of different standards and/or specifications, dealing with different aspects of the interoperability problem such as content packaging, content communication and metadata descriptions. The first effort towards addressing the full spectrum of interoperability issues between e-learning systems was Sharable Content Object Reference Model (SCORM), by ADL[1]. SCORM was widely accepted and endorsed by the e-Learning community and managed to achieve great success with version 1.2 [HYPERLINK \l "htt2" 1]. The following version, however, failed to continue that trend. SCORM 20042]**}**, with the addition of learning content sequencing capabilities increased in size and complexity and the adoption rate was negatively impacted.

[1] http://www.adlnet.gov

1.1 Motivation

SCORM has attracted considerable attention from the e-Learning community since its inception. It was hailed as the specification that would bring order and harmony to the chaotic world of e-Learning and gave rise to great expectations. These expectations were partially fulfilled with version 1.2, which managed to achieve great success, as it was quickly adopted by both government and industry. However, SCORM 1.2 lacked one very essential feature: it didn't define a method to control the sequencing of content objects. Consequently, content developers had to create each course as one monolithic SCO (Sharable Content Object) with embedded custom sequencing logic to control how learners navigate through the content. This hindered the modularity and reusability of SCOs' as their size didn't allow them to be reused and repurposed in different contexts. ADL recognized this problem and added a sequencing and navigation model in the subsequent version, SCORM 2004. This addition enabled content designers to represent the intended behavior of an authored learning experience such that any SCORM compliant LMS can consistently realize it, by properly sequencing content objects. The sequencing logic is placed external to the content objects, thus a content developer can create granular, reusable SCOs as they were intended to be. The SCORM Sequencing and Navigation model relies on the *IMS Simple Sequencing* specification [HYPERLINK \l "IMSS" 3] which, despite its name, is complex, impalpable and difficult to implement. ADL does not define levels for LMS compliance, thus LMS vendors must implement SCORM as a whole to become SCORM compliant. This greatly affected the SCORM adoption rate and even widespread LMSs have not yet accomplished compliance to SCORM 2004. In addition, it was observed that LMS vendors tend to interpret SCORM requirements differently. Therefore, ADL added more than 75 new test cases to the SCORM 2004 4th Compliance Test Suite to ensure stricter adherence to SCORM requirements. This will improve the interoperability across LMSs but the amount of work for implementers to achieve SCORM compliance is increased. Currently, there are no known SCORM 2004 4th Edition implementations as the ADL Certification Program has not yet begun.

1.2 Thesis Objectives

This thesis aims to provide a reference implementation of the latest version of the Sharable Content Object Reference Model (SCORM 2004 4th Edition) grounded in a fully functional pre-existent learning management system. For the purpose of this thesis we selected the learning management system *eFront*[2] which is used in the course "*CS-100 Introduction to Computer Science***}**" offered by the Computer Science Department of the University of Crete. Our goal is to accomplish a fully compliant implementation, according to the requirements defined in the *SCORM 2004 4th Edition Testing Requirements (TR) Version 1.1* [4], published by ADL. In addition, we attempt to propose an architecture that is transparent to the learner in order to provide an overall better and more effective learning experience. The implementation is evaluated using *SCORM 2004 4th Edition Version 1.1.1 Test Suite*[3]. Finally, we compare the current version of SCORM with its highly successful predecessor, SCORM 1.2, and judge whether the new features and additions justify the increased complexity.

1.3 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 provides the reader with a general introduction to the field of e-Learning. It begins with a brief description of the e-Learning concept and a description of the term of "*Learning Object*" which is fundamental not only in e-Learning but also in learning per se. Afterwards, a basic understanding of the e-Learning standardization landscape is provided, which includes a classification of e-Learning standards, the most important standardization bodies and their interrelationship, as well as the most successful standardization efforts. Chapter 3 analyzes the "*Sharable Content Object Reference Model" (SCORM*), on which this thesis is based on. It begins with an overview of SCORM and its evolution since its inception and then moves on its main constituting parts: The Content Aggregation Model, the Run-Time Environment and the Sequencing and Navigation. Chapter 4 describes our proposed architecture for integrating the latest version of the SCORM (SCORM 2004 4th Edition) into a pre-existent learning management system. The proposed architecture was successfully implemented in the *eFront* learning management system and the corresponding implementation details are provided. The chapter concludes by describing how we confirmed the validity of our work against SCORM

[2] http://www.efrontlearning.net/

[3] http://www.adlnet.gov/Technologies/scorm/SCORMSDocuments/2004%204th%20Edition/Test%20Suite.aspx

conformance requirements. Finally, Chapter 5 holds conclusions drawn from this thesis and outlines potential topics for future work.

Chapter 2: **E-Learning & Standards**

This chapter provides a general introduction to the field of e-Learning. Starting with a brief description of the e-Learning concept, we will then introduce the term “Learning Object” by presenting several definitions that have been posited in literature, in an attempt to clarify the term. Afterwards, a basic understanding of the e-Learning standardization landscape is provided, which includes a classification of e-Learning standards, the most important standardization bodies and their interrelationship, as well as the most successful standardization efforts.

2.1 E-Learning

"*E-Learning is just-in-time education integrated with high velocity value chains. It is the delivery of individualized, comprehensive, dynamic learning content in real time, aiding the development of communities of knowledge, linking learners and practitioners with experts*" [5]. While this definition is pretty much accurate, it is important to clarify that e-learning aims to be integrated to the actual learning process and not act as a simple aid. E-learning brings revolutionary ideas in the classic learning methodologies. In the conventional learning process, a student is obliged to be physically present at a specific place and time, remain for a predefined time period and must abide to several rules. These restrictions make learning a time and place bound process, constrained and not appealing to the learners. In contrast, in an e-Learning environment, the student has a relative freedom to determine the course schedule, which can dynamically change and, as long as a web browser and an internet connection are available, there are no location restrictions. In addition, new possibilities emerge in an e-learning environment. While the concept of individualized instruction is rather old, it has only become a feasible possibility in recent years, with the advances in the field of e-Learning. Print media cannot support such types of instruction and instructor-led courses are of great cost. Self-paced learning is another feature offered in an e-Learning and not available in a conventional classroom. Advanced learners can speed through or even bypass instruction which is below their level while novices slow down to their own pace and avoid frustration. Last but not least, e-Learning environments can offer personalized learning which is a unique educational model that is dynamically adapted to the needs and preferences of each individual learner. Personalized learning boosts self-motivated, independent, and self-directed learning taking under consideration four important factors [6] [7]:

- the influence of a comprehensive set of key psychological factors (co-native, affective, social, and cognitive) that influence learning differently
- the often overlooked dominant impact of emotions and intentions on learning,
- critical human relationships between learning environments, key psychological sources that influence learning differently, and learning ability
- new ways to design supportive learning environments that individually adapt to how people want to learn, from the more comprehensive perspective

There are two types of e-learning, synchronous and asynchronous. The distinction is made on the basis of the communication model that is used. Synchronous learning typically involves the instructor broadcasting live video or audio in parallel with a slideshow presentation manually controlled by him. The learners are able to pose questions and request comments through a chat. Asynchronous learning includes email, mailing lists, forums, wikis and blogs and is characterized as self paced because of the way it is offered.

2.1.1 Learning Object

The explosive growth of the Internet in size, usage and applications has greatly affected the way in which learning material is designed, developed and delivered to the learners. An instructional technology, called learning objects, has played a significant role in this change. The term was introduced by Wayne Hodgins in 1994 when he named a CedMA[4] (Computer Education Management Association) working group "*Learning Architectures, APIs and Learning Objects*". Since then, many definitions for the term "learning object" have been posited in the literature and, as Wiley [8] observes, "*the proliferation of definitions for the term 'learning object' makes communication confusing and difficult*".

Before we dive into the definitions, let us illustrate the concept of learning objects using a metaphor. It is a common practice to use metaphors for the description of technological concepts. The most prevalent one for the case of learning objects is the Lego metaphor [9]. As we know, although Lego blocks come in many shapes and sizes, they are highly reusable and can be easily stacked together to form various constructions. Similarly, learning objects technology can be used to break down instructional content into the smallest useful pieces of instruction capable of being assembled into as well as reused in larger instructional

[4] http://www.cedma.org/

structures. Wiley [8] criticizes the Lego metaphor for being over-simplistic and potentially misleading. As he states, consider the following properties of a LEGO block:

- Any LEGO block is combinable with any other LEGO block.
- LEGO blocks can be assembled in any manner you choose.
- LEGO blocks are so fun and simple that even children can put them together.

Since the metaphor initially fits well, the above properties are also considered as properties of learning objects. Wiley [8] concludes that "*a system of learning objects with these three properties cannot produce anything more instructionally useful than LEGOs themselves can. And if what results from the combination of learning objects is not instructionally useful, the combination has failed regardless of whatever else it may do."* In our opinion, the metaphor stands well, even in this case. Neither every combination of Lego blocks is meaningful, nor can everyone produce really good results. In the rest of this subsection, several definitions of the term "learning object" are presented and discussed, as we try to capture the essence of the concept.

The Learning Technology Standards Committee[5] (LTSC) defines a learning object as "*any entity, digital or non-digital, which can be used, re-used or referenced during technology supported learning. Examples of technology supported learning include computer-based training systems, interactive learning environments, intelligent computer-aided instruction systems, distance learning systems, and collaborative learning environments. Examples of Learning Objects include multimedia content, instructional content, learning objectives, instructional software and software tools, and persons, organizations, or events referenced during technology supported learning.*" Although this definition provides some guidance, it is so broad and over-inclusive that almost everything falls under it. For example, a keyboard or a Learning Management System (LMS) can serve a useful purpose during technology supported learning, but they certainly cannot be characterized as learning objects. Wiley [8] suggests the following narrower definition: "*Learning object is any digital resource that can be reused to support learning*". Likewise, Weller [10] defines learning object as "*a digital piece of learning material that addresses a clearly identifiable topic or learning outcome and has the potential to be reused in different contexts*". These definitions differ from LTSC's definition in two ways. First, they clearly exclude anything non-digital and non-reusable. Actual people, electronic devices, textbooks or other traditional educational material are left

[5] http://www.ieeeltsc.org

out. Second, according to these definitions, a learning object must support the learning process rather than simply be used during. An alternative definition is provided by Koper [11]: "*Learning object is any digital, reproducible and addressable resource used to perform learning activities or learning support activities, made available for others to use*". In this definition, a new criterion is added, that of addressability, which cuts off resources that are not connected with metadata and a URL for discovery and access. In addition, Koper [11] classifies learning objects into the following categories:

- **Knowledge objects** are learning objects which contain information for people to learn from or to use while supporting the learning activities of others (for example teachers with students). An example is a web page with a series of information objects to learn, (e.g. about sensory systems); or a teachers' manual.
- **Tool objects** are learning objects to learn with or to use while supporting the learning activities of others. Examples include electronic tools (Java applets for statistics) or simulations.
- **Monitor objects** are objects which provide information about the learning progress and tracking of one's own learning or that of others.
- **Test objects** are learning objects used to assess learning results, learning progression or prerequisites (e.g. a complete test, a test item).
- **Resource organization** objects are learning objects at a lower level that enable the resources to be organized in a particular way. Examples include aggregating pictures and text to a paragraph and paragraphs to sections and sections to chapters. These arrangements of objects, or "organizations", are used as the information reference containers in higher level objects such as knowledge objects, test objects and activity descriptions.

It becomes apparent that the term "Learning Object" is ambiguous. There are numerous definitions available and each one represents the primary interests or concerns of its proponents [12]. However, a consensus has been established in the field of e-Learning about the functional requirements of learning objects which should be accessibility, reusability and interoperability.

2.2 E-Learning Standards

A standard is defined by the National Standards Policy Advisory Committee [13] as "*a prescribed set of rules, conditions, or requirements concerning definitions of terms; classification of components; specification of materials, performance, or operations; delineation of procedures; or measurement of quantity and quality in describing materials, products, systems, services, or practices.*" It is unarguable that we are living in the era of standards. Units of measurement, paper formats, the size of railroad tracks, Wi-Fi protocols and the format of credit cards, to name a few, are all based on standards. These are only a few examples, but enough to exhibit the crucial role of standards in our everyday lives. However, unless problems arise, they usually go unnoticed. Problems may occur when standards are poorly designed, inconsistently implemented, not globally adhered, or completely missing. For example, there is no globally adhered standard for electrical plugs and socket-outlets. As a result, electrical plugs and socket-outlets differ by country and by region in shape, size and type of the connector, causing many inconveniences to world travellers. Currently, e-Learning is shifting from the chaotic, lawless “no standards” state to a more organized and stable “with standards” state. In the following, after gaining a basic understanding about the different types of e-Learning standards and their purpose, the most important standardization bodies as well as the most successful standardization efforts are presented.

2.2.1 Types of eLearning Standards

The key issues regarding interoperability of eLearning tools and technologies are content description (metadata), content packaging, learner management and communication of the learning process results [14] [15]. Below, we discuss these issues and enumerate the most imortant standardization efforts for each one.

Content Description (Metadata)

Learning content and catalogue offerings must be labelled in a consistent way to support the indexing, storage, discovery (search), and retrieval of learning objects by multiple tools across multiple repositories. The foundation of content description is metadata which attempt to describe the content, format, purpose and structure of data. In the context of eLearning, they are referred to as learning object metadata. The most important standardization efforts dealing with content description are listed below:

- The IEEE Standard for Learning Object Metadata(LOM) [16]

- The Dublin Core Metadata Element Set [17]

Content Packaging

Content packaging specifications and standards allow learning content to be wrapped into self-contained packages and transported between different repositories and systems. This enables learning content to be created by one tool, modified by another tool, stored in a repository maintained by one vendor, and used in a delivery environment produced by a different vendor. Content packages include: a) raw content (HTML files, images, video, audio, Flash, Shockwave, JAVA applets, PDF files), b) assembly, delivery and presentation information. Several initiatives are creating content packaging specifications and standards:

- The IMS Content Packaging specification [18]
- The *IMS Simple Sequencing* specification [3]
- Aviation Industry CBT Committee guidelines and recommendations for computer managed instruction [19]

Learner Information

Learning Information standards addresses the interoperability of learner information across multiple system components. Learner profile information can include personal data, learning plans, learning history, accessibility requirements, certifications and degrees, assessments of knowledge (skills/competencies), and the status of participation in current learning. The most important efforts to standardize Learner information are:

- The IMS Learner Information Package (LIP) specification [20]
- The Schools Interoperability Framework(SIF) [21]

Content Communication

Learning content, once launched, must have a way to exchange data with the LMS (i.e. learner progress, performance, preferences etc). The most widely accepted communication models are:

- The IEEE Standard for ECMAScript Application Program Interface (API) for Content to Runtime Service Communication [22]
- The AICC CMI001 Guidelines for Interoperability [19]

2.2.2 Value of e-Learning Standards

It is common belief that standards will drive e-Learning towards maturity. The impact of elearning standards is manifold. "*Not only would the development and use of international standards produce a direct cost savings, but the information technology systems could be used in a wider range of applications, and used more efficiently. Better, more efficient and interoperable systems, content, and components will produce better learning, education, and training – which has a positive effect upon all societies*" [23]. This rather optimistic view has to be analyzed further in order to realize the true potential of eLearning standards. To begin with, software tool vendors will not have to develop multiple interfaces to communicate with different and often heterogeneous systems, resulting in lower development costs and an increase of the size and scope of the potential market. This is critical because there is an increasing demand for the integration of learning services into systems such as ERP (Enterprise resource planning) and CRM (Customer relationship management*)* [24] [25]. Learning content producers will focus on the development of better learning content instead of putting effort into producing different versions of the same content for different platforms and applications. Instructors will be able to effortlessly find, repurpose, combine and reuse existing learning content resulting in cost and time efficiencies. In addition, they will be able to shift between tools and platforms and find those that better satisfy their needs with minimal transition cost. Learning environment developers will be sure that a wide variety of content works on their systems without complications. They will not have to persuade or hire content authors to develop content specifically for their platform [14]. As far as learners are concerned, it is expected that all the aforementioned effects of eLearning standardization will ultimately result to better tools and applications, and learning content.

2.2.3 Standardization Bodies

Several committees, consortiums, working groups, and other bodies are involved with the development and the promotion of e-Learning standards. Those attracting the most interest from the e-Learning community currently are presented in the following sections.

2.2.3.1 Instructional Management Systems (IMS) Global Learning Consortium

The IMS Global Learning Consortium[6] (usually referred as IMS GLC) is a global, non-profit organization which actually serves as an umbrella organization inside of which numerous e-Learning standards organizations co-exist. At present, it is supported by over 135 organizations and encompasses over 30 working groups, domestically and internationally, each working on different interoperable specifications and standards. IMS was born in 1995 as a project of the National Learning Infrastructure Initiative of EDUCAUSE[7] to create a set of widely adopted standards for exchanging learning content in higher education. Over the years, IMS's activities range has expanded to cover most of the data elements used in distributed and collaborative learning in various learning environments including K-12[8], higher education, corporate and government training. The specifications concern both online (web based leaning management systems) and offline settings (instructional material that can be delivered on a CD-ROM or DVD) in which learners and instructors may be separated not only in space but also in time. In particular, IMS is focusing on the development of XML-based specifications to capture the key characteristics of courses, lessons, assessments, learners and groups. Along with these specifications it publishes Best Practices Guidelines to provide a structure for representing eLearning meta-data which aid in clarifying and finding learning content. The most widely spread IMS specifications are IMS Question and Test Interchange [26], IMS Content Packaging [18] and IMS Simple Sequencing [3]. The IMS Content Packaging and the *IMS Simple Sequencing* specifications are thoroughly described in the next chapter as part of the Sharable Content Object Reference Model.

IMS Question and Test Interoperability

The IMS Question and Test Interoperability (QTI) specification, as the name suggests, addresses the interoperability problem between assessments and their corresponding results and heterogeneous item banking, test authoring tools, test delivery systems,

[6] http://www.imsglobal.org
[7] http://www.educause.edu
[8] K–12 is a designation for the sum of primary and secondary education. It is used in the United States, Canada, and some parts of Australia.

Learning Management Systems and data management systems. The specification describes a sufficient number of data models to represent a wide diversity of assessment material. It supports all commonly used question types multiple-choice, essay, likert-scale[9], and fill-in-the-blank which can be constructed using a core set of presentation and response structures, and the results can be collected and scored using a variety of methods. The IMS QTI specification contains two main components, the Assessment-Section-Item-Information (ASI) Model and the Results Reporting Model, plus a lightweight version of the QTI specification for introductory implementation, the IMS QTI-Lite Specification. The QTI ASI Information Model defines the following data structures:

- Item(s): The smallest self-contained unit that can be exchanged within QTI. It contains the Question, presentation and rendering instructions, the processing to be applied to the user's responses(s), the feedback that may be presented to the user such as hints and possible solutions and metadata describing the Item.
- Sections(s): They group item series and/or other sections in a meaningful way. A section supports the following needs: An educational paradigm may dictate grouping questions by subtopic or difficulty and obtain the score for each group separately. If this is the case, different Sections can be used to represent different subtopics or difficulty levels. Furthermore, the questions order may be significant for an educational paradigm. Sections define which selection and ordering algorithms will be applied to the contained Sections and/or Items and can help limit the scope of the sequencing and ordering instructions
- Assessment: It is equivalent to a test and one and only one Assessment can be included in a QTI-XML instance. There are no relationships which can be defined between different Assessments and each Assessment must contain at minimum one Section. An Assessment cannot hold Items directly underneath it. It defines which selection and ordering algorithms will be applied to the contained Sections, the group evaluation process to produce the overall score and the corresponding feedback.
- Object Bank: It is a bundle of Items, Sections or a mixture of Items and Collections. An Object Bank contains metadata which enable its contents to be searched. It can be used to transfer content between systems or it can act as a database of objects from which Assessments can be composed.

[9] Likert scaling is a bipolar scaling method, measuring either positive or negative response to a statement (i.e. Strongly disagree – Disagree - Neither agree nor disagree – Agree -Strongly agree)

The Results Reporting Model defines the following data structures:

- Item_result: The detailed assessment information of a particular instance of an Item.
- Section_result: The detailed assessment information of a particular instance of a Section.
- Assessment_result: The detailed assessment information of a particular instance of an Assessment.
- Summary_result: It contains a generic summary result of a particular instance of a single evaluation (e.g. an assessment, section or item).
- Context: It contains the contextual information concerning the actual result being reported e.g. the name of the participant (in an agreed formatted style), relevant dates (the date after which the validity of the results expire).
- Result: It contains the set of results relevant to an actual attempt of an assessment or some other form of evaluation in conjunction with information describing the conditions under which the evaluation was taken. It can contain multiple results for a single participant, multiple results for a single test undertaken by several participants as well as any combination of the above two.

2.2.3.2 Aviation Industry CBT Committee

Aviation Industry CBT Committee[10] (AICC) is an international association of aircraft manufacturers, aviation trainers (military, commercial, and civilian), government and regulatory agencies, e-learning tools vendors, and e-Learning courseware developers. It was formed in 1988, to support the need for hardware standardization of CBT (Computer Based Training) delivery platforms in the aviation industry. It seems quaint by today's standards, but back in the early 1980s CBTs were not only developed using proprietary software but also proprietary hardware. Thus, in 1989, AICC published common platform guidelines for CBT delivery. The next standardization effort by AICC was a DOS-based digital audio guideline before the advent of Windows-based multimedia standards. The guideline enabled end-users to use a single audio card for multiple vendors' CBT courseware and is still used by many older legacy CBT applications. In 1993, the AICC published CMI001 - AICC/CMI Guidelines for Interoperability which is generally acknowledged as the first runtime interoperability specification for CMI (Computer Managed Instruction) Systems, more

[10] http://www.aicc.org/

commonly known as Learning Management Systems (LMS). This specification was originally designed to share data with LAN-based CBT courseware from multiple vendors and was updated in January 1998 to add a web-based interface called HACP. HACP stands for Hypertext AICC Communications Protocol and uses HTTP form posts to exchange data with the LMS. In September 1999, the CMI001 specification was updated once more to include a JavaScript API runtime interface. This addition is the basis of the SCORM runtime environment. At last, in 2005, AICC published The Package Exchange Notification Services (PENS) guideline which aims to simplify the deployment of content in LMSs as long as both conform to AICC or ADL SCORM.

2.2.3.3 IEEE Learning Technology Standards Committee

The Institute of Electric and Electronic Engineers[11] (IEEE) is a global, non profit, technical professional organization with more than 365.000 individual members in around 175 countries. One of the main activities of IEEE is developing standards, a duty delegated to the IEEE Standards Association (IEEE-SA). IEEE-SA, with a portfolio of 900 active standards and more than 400 standards in development, is one of the leading standards-setting organizations in the world. IEEE standards concern a wide variety of areas, including among others, Aerospace Electronics, Bioinformatics, Communications, Instrumentation and Measurement, and Information Technology. The IEEE Learning Technology Standards Committee (LTSC), under the aegis of IEEE standards Association (IEEE-SA), develops internationally accredited technical standards, recommended practices, and guides for learning technology. The term “Learning technology”, as it is used here, includes software components, tools, technologies, and design methods that facilitate development, deployment, maintenance, and interoperation of computer-based education and training components and systems. IEEE LTSC often cooperates with other standards development organizations (SDOs) in the educational domain such as AICC and IMS. The most widely spread IEEE LTSC standards are:

- The IEEE Standard for Learning Object Metadata [16]
- The IEEE Standard for Learning Technology - Data Model for Content to Learning Management System Communication [27]
- The IEEE Standard for ECMA Script Application Program Interface (API) for Content to Runtime Service Communication [22]

[11] http://www.ieee.org

These standards have been adapted for use in SCORM and are thoroughly described under its context in the next chapter.

2.2.3.4 Advanced Distributed Learning

The Advanced Distributed Learning[12] (ADL) Initiative was launched in 1997 with primary mission to harness the power of learning and information technologies in order to standardize and modernize education and training of the U.S armed forces. Capstone of these efforts was the publication of Sharable Content Object Reference Model (SCORM) in 1999 [2]. Although it was envisaged by the U.S Department of Defence, SCORM is being developed through active collaboration between the US federal government, industry and academia. In order to facilitate this collaboration, the ADL established the ADL Co-Laboratory Network[13] which provides a collaborative environment for harmonizing learning technology research, integration and assessment into a unified ADL strategy. SCORM at a high-level is a collection of specifications and standards which boost, accessibility, interoperability, durability and reusability in content and systems. It is considered the most robust and widely adopted standardization effort and it will be thoroughly discussed in the following chapter.

[12] http://www.adlnet.gov
[13] http://www.adlnet.gov/About/Pages/CoLabNetwork.aspx

Chapter 3: **SCORM**

SCORM (Sharable Content Object Reference Model) is the capstone of ADL Initiative efforts to provide access to the highest quality learning and performance aiding, which can be tailored to individual needs and delivered cost-effectively anytime and anywhere [2]. The "*Sharable Content Object*" part of the acronym denotes that SCORM focus on the creation of content objects that can be shared across different repositories and systems. The "*Reference Model*" indicates that SCORM isn't actually a standard but a reference model. When the ADL Initiative started, many specifications and standards already existed in the e-Learning landscape. However, these standards were isolated and dealt with different parts of the interoperability problem. ADL Initiative recognized this issue and instead of producing more, possible overlapping standards, directed their efforts towards the extension and harmonization of existing standards. ADL [2] describes SCORM as a "*reference model that integrates a set of inter-related technical standards, specifications, and guidelines designed to meet high-level requirements for learning content and systems*". The high-level requirements for all SCORM-compliant[14] e-learning environments, known as SCORM "*ilities*", are:

- **Accessibility:** The ability to locate and access instructional components from one remote location and deliver them to many other locations
- **Interoperability:** The ability to take instructional components developed in one location with one set of tools or platforms and use them in another location with a different set of tools or platforms.
- **Durability:** The ability to withstand technology evolution and changes without costly redesign, reconfiguration or recoding.
- **Reusability:** The ability to incorporate instructional components in multiple applications and contexts.

Moreover, SCORM is in sync with the eLearning industry and research, as it silently follows the "*Web-enabled assumption*". The "*Web-enabled assumption"* asserts that the Web provides the best opportunity to maximize access to and reuse of learning content. The latest SCORM release consists of five distinct "books":

[14] There is confusion between the terms "conformance" and "compliance". When we are referring to standards the appropriate term to use is "compliance". Thus, because several specifications in SCORM documentation have been accepted as IEEE standards, the right term to use is compliance.

- **SCORM Overview** which contains high-level conceptual information, the history, current status and future direction of ADL and how the other books relate. It also contains an overview of the Test Suite and the Runtime Environment, which ADL provides complementary to the SCORM document suite.
- **Content Aggregation Model (CAM)** which describes the types of content objects used in a learning experience, how to package these content objects for exchange among different systems and repositories, how to describe these content objects using metadata to enable search and discovery and how to define different sequencing strategies using properly formatted sequencing rules.
- **Run-Time Environment (RTE)** which describes the Learning Management System (LMS) requirements in managing the run-time environment (i.e. content launch process, standardized communication between content and LMSs and standardized data model elements used for passing information relevant to the learner's experience with the content).
- **Sequencing and Navigation (SN)** which describes the available navigation requests (issued either by the learner or the LMS) and how these navigation requests are interpreted by a SCORM compliant LMS.
- **Compliance Requirements** which contains a detailed list of the compliant requirements that an LMS must adhere to be SCORM compliant. These requirements are verified by the ADL SCORM compliance test suite.

In the following sections, the evolution of SCORM since its inception is illustrated. Afterwards the main parts that constitute SCORM are presented, as defined in the latest SCORM release (SCORM 2004 4th Edition):

- The Content Aggregation Model
- The Run-Time Environment
- The Sequencing and Navigation.

3.1 History of SCORM

In January 1999 an executive order [28] tasked ADL to develop consensus standards for training software and associated services. Prior to 1999, many organizations were drafting a variety of different standards and/or specifications, which were focusing on different aspects of e-Learning systems. Thus, rather than starting from scratch, ADL decided to create a practical profile of existing specifications and standards [**29**]. In standardization, a profile is an agreed-upon subset and interpretation of a specification or standard. During the process, some refinements and additions were necessary and the end result was a set of "books", the so-called SCORM.

The first version of SCORM (then Sharable Courseware Object Reference Model) was released in January 2000. Initial contributions came from the *AICC/CMI CMI001 Guidelines for Interoperability*, which includes the *AICC Course Structure format* and the *AICC CMI Data Model*, and the *IMS Learning Resource Meta-data specification*. This version actually served as a proof of concept and it signaled a test and evaluation phase during which researchers and early adopters were encouraged and expected to develop trial implementations based on these specifications. The experience gained during this phase was the key driver towards the next version of SCORM. One year later, in January 2001, ADL released SCORM Version 1.1. This was the first production version of SCORM and it included numerous corrections and improvements based on the feedback received from the initial release. Moreover, during the test and evaluation phase of the SCORM Version 1.0, representatives of the *IEEE Learning Technology Standards Committee (LTSC)* and the *AICC* decided to remove a significant number of elements both from the *AICC Course Structure format* and the *AICC CMI Data Model.* ADL followed their decision and deprecated those elements in SCORM version 1.1. Consequently, the amount of work and maintenance for implementers was reduced. Finally, it is worth mentioning that the acronym SCORM was changed to mean "Sharable *Content Object Reference Model*". In the same year, ADL released SCORM version 1.2. This version introduced a major technical change, the inclusion of IMS Content Packaging specification. This change is a milestone on SCORM's evolution because content packaging is a prerequisite for meeting one of its high level requirements, interoperability. Although SCORM version 1.2 is now replaced by SCORM 2004, it still remains the most popular standard in e-learning industry. In January 2004, ADL released the 1st Edition of the highly anticipated SCORM 2004. Highlight of this version was the addition of learning content sequencing capabilities as defined by the *IMS Global Learning Consortium's Simple*

Sequencing (SS) specification to address the need for dynamic presentation of learning content based on learner performance. In addition, since the initial release of SCORM, certain specifications and standards incorporated in SCORM have evolved. Thus, SCORM was updated to reflect the changes made. These changes, along with many others derived from community feedback, was summarized under the document entitled *"SCORM Version 1.2 to SCORM 2004 Changes"*, published by ADL. The next version of SCORM (SCORM 2004 2nd Edition), released in July of the same year, primarily resolved defects found during SCORM adoption. However, many issues continued to rise and later on ADL published *"SCORM 2004 2nd Edition Addendum"* which described the actions needed to be taken by implementers to address those issues. The next version of SCORM, SCORM 2004 3rd Edition, was released in October 2006. It contained enhancements and corrections identified by the community, as well as requirements and practices associated with LMS provided user interfaces devices. Until then, it was left to the LMS to provide a consistent and appropriate user interface. The next version of SCORM (SCORM 2004 4th Edition) was published in March 2009, but it is not yet officially released. It contains further clarifications on concepts and requirements related to the sequencing specification and enhancements that ease the creation of sequenced content. In addition, it describes extensions to the *IMS packaging specification* that allow sharing more data between SCO's than the previous versions.

The SCORM Content Aggregation Model (CAM) book describes the instructional components used in a learning experience and offers information on:

- how to package them for exchange among different systems
- how to describe them using metadata to enable search and discovery
- how to apply different sequence strategies to enrich the learning experience

It is divided into four parts: Content Model, Content Packaging, Meta-data, and Sequencing and Presentation.

3.1.1 Content Model

The Content Model is the conceptual foundation of SCORM (Figure 3-1). It formally describes the SCORM components used to build a learning experience from learning resources. Moreover, it facilitates the reusability of learning content, one of the high level requirements of SCORM, by defining different aggregation levels for the learning resources. The SCORM Content Model consists of assets, sharable content objects (SCOs), activities, content organizations and content aggregations.

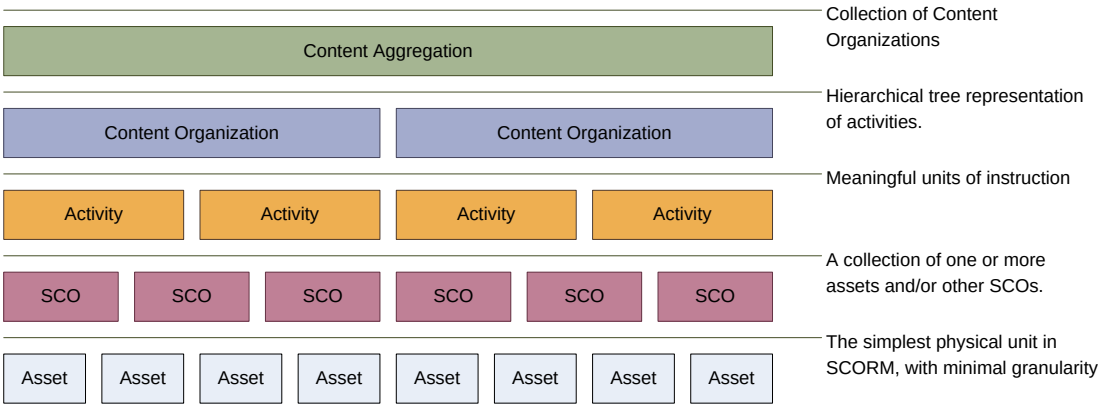


Figure 3-1: SCORM Content Model Components

To begin with, an asset is the simplest physical unit in SCORM, with minimal granularity. An asset is any file that can be delivered through a regular Web browser, such as a static HTML page or a PDF document. Although assets can be launched as standalone units of instruction, they are more commonly used as part of higher-level aggregations (e.g. SCOs).

A SCO is a collection of one or more assets and/or other SCOs. SCOs, unlike assets, can communicate with the runtime environment in which they are launched. Each SCO must at least be able to initialize and terminate a communication session with the LMS by calling the corresponding methods of the SCORM Runtime Environment. SCOs are intended to be fully portable, thus SCOs cannot incorporate sophisticated content that requires specific advanced server functionalities in order to run properly or reference other SCOs. SCOs and assets are the two types of reusable learning objects used in SCORM. SCORM does not impose any particular constraints on the size of a SCO. However, content designers should take under consideration that the SCO represents the lowest level of granularity that can be tracked by an LMS and the size of each SCO greatly affects its ability to be repurposed and reused in different contexts.

An activity is a meaningful unit of instruction, with a specific learning intent, that may directly provide learning resources (either assets or SCOs) or be composed of several other activities. There is no limit to the number of levels of nesting for activities. If an activity consists of other activities it is referred to as a cluster activity. It is worth mentioning that sequencing can only be applied to activities.

A content organization is a hierarchical tree representation of activities that shows how they relate to one another. Finally, a content aggregation is a container for content organizations

and the term is often used to describe the content package which is used to transfer learning content between systems or to store it in a repository.

3.1.2 Content Packaging

Content packaging allows learning content to be wrapped into self-contained packages and transported between different repositories and systems. SCORM 2004 relies on the IMS Content Packaging specification to provide this capability. A content package consists of an XML document called the manifest file (imsmanifest.xml) and the actual content (i.e., physical files).

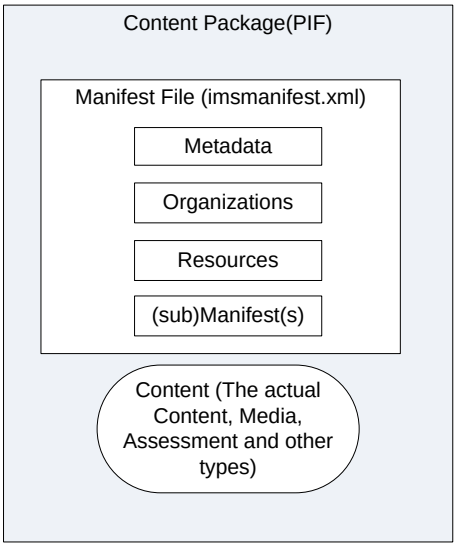


Figure 3-2: Content Package Conceptual Diagram

The manifest file describes the content structure and the associated resources of the package. The top level element in the imsmanifest.xml file is the <manifest> element (Listing 3-1). It is a parent element, meaning that no values are associated with it. The identifier attribute identifies the manifest, while the version attribute is used to distinguish manifests with the same identifier. Finally, ADL suggests all the namespace declarations to be included in the <manifest> element.

```
<manifest identifier = "LMSTestPackage_CM-01" version = "1.1"
    xmlns = "http://www.imsglobal.org/xsd/imscp_v1p1"
    xmlns:adlcp = "http://www.adlnet.org/xsd/adlcp v1p3"
    xmlns:adlseq = "http://www.adlnet.org/xsd/adlseq v1p3"
    xmlns:adlnav = "http://www.adlnet.org/xsd/adlnav v1p3"
    xmlns:imsss = "http://www.imsglobal.org/xsd/imsss"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation = "http://www.imsglobal.org/xsd/imscp_v1p1 imscp_v1p1.xsd
                          http://www.adlnet.org/xsd/adlcp v1p3 adlcp v1p3.xsd
                          http://www.adlnet.org/xsd/adlseq v1p3 adlseq v1p3.xsd
                          http://www.adlnet.org/xsd/adlnav v1p3 adlnav v1p3.xsd
                          http://www.imsglobal.org/xsd/imsss imsss_v1p0.xsd">
```

```
<!-- imsmanifest contents -->
</manifest>
```

Listing 3-1: Example of a <manifest> element

The <manifest> consists of the meta-data, organizations and resources elements. The metadata element contains metadata describing the manifest (Listing 3-2). While the metadata here describe the content package as whole, metadata can be inserted into various locations within the manifest to describe different parts of the content package.

```
  <metadata>
     <schema>ADL SCORM</schema>
     <schemaversion>2004 4th Edition</schemaversion>
     <adlcp:location>packageMetadata.xml</adlcp:location>
  </metadata>
```

Listing 3-2: Example of a <metadata> element

The value of the schema is fixed to ADL SCORM and denotes that the Content Package adheres to the schema defined by SCORM. The <schemaversion> indicates the version of the schema. SCORM allows metadata to be inserted either inline, inside the metadata element, or in a separate XML file. If the metadata reside in a separate file, the <adlcp:location> is used to specify its location. SCORM highly recommends metadata to conform to the *IEEE Learning Object Metadata (LOM)* standard in order to ensure that they will be understandable by intelligent agents. Metadata are discussed further in the following section. The organizations element represents the content aggregation component of the SCORM content model (Listing 3-3) .The manifest must contain a single organizations element.

```
  <organizations default = "CM-01">
      <organization identifier = "CM-01">
         <title>LMS Test Content Package CM-01 </title>
         <item identifier = "activity 1" identifierref = "SEQ01" parameters = "?tc=CM-
01&amp;amp;act=1">
            <title>Activity 1</title>
         </item>
         <item identifier = "activity 2" identifierref = "SEQ02" parameters = "?tc=CM-
02&amp;amp;act=2">
            <title>Activity 2</title>
         </item>
      </organization>
  </organizations>
```

Listing 3-3: Example of an <organizations> element

The organizations element can contain one or more organization elements. The organization element represents the content organization component of the SCORM content model. It contains one or more item elements that can be nested to any depth. The item element represents the activity component of the SCORM content model. Each <item> has a title that

is typically used by the LMS in the table of contents of the content package. A leaf item (an item with no children) may reference a resource (a SCO or an asset). The resource that it is referenced, if any, is specified by the identifierref attribute. Presumably, each <resource> has an identifier attribute. Resources (Listing 3-4) may be referenced more than once.

```
<resources>
      <resource identifier="SEQ01" type="webcontent" adlcp:scormType="sco"
href="SequencingTest.htm" xml:base="resources/">
         <file href="SequencingTest.htm"/>
      </resource>
      <resource identifier="LMSFNCTS01" type="webcontent" adlcp:scormType="asset">
         <file href="common/lmsrtefunctions.js" />
      </resource>
      <resource identifier="JAR01" type="webcontent" adlcp:scormType="asset"
xml:base="common/">
         <file href="LMSTest.jar" />
      </resource>
      <resource identifier="ABOUT01" type="webcontent" adlcp:scormType="asset">
         <file href="common/About.js" />
      </resource>
<resources>
```

Listing 3-4: Example of a <resources> element

The adlcp:scormType defines whether the resource is SCO or asset. Each resource has an href attribute which is used by the LMS to locate and launch the resource. The value of the href attribute is a Uniform Resource Locator (URL) and it is relative to the location of the imsmanifest.xml file. Optionally, the xml:base attribute may provide a relative path offset for the files contained in the manifest. The resource element contains a listing of the files required to launch the resource. Each file is specified by a file element with an href attribute. The value of the href specifies the location of the file, relatively to the imsmanifest.xml file.

The content package can be delivered either in a simple directory or in the form of a compressed archive file. In the latter case, SCORM recommends content packages be created as PIFs (Package Interchange File) which is a concise Web delivery format. If the PIF is selected for delivering the content package, SCORM requires that the PIF be conformant with RFC 1951 and the archive format be PKZip v2.04g (.zip).

3.1.3 Metadata

In order to describe the learning content, SCORM recommends the *IEEE Learning Object Metadata (LOM)* standard along with the *IEEE Standard for Extensible Markup Language (XML) Binding for Learning Object Metadata Data Model*. The manifest provides the means for associating metadata with each of the defined levels of content granularity. Thus, SCORM metadata can be divided into:

- **Content Aggregation Metadata:** Content Aggregation Metadata describes the content aggregation (i.e., the content package) as a whole. The purpose of applying Content Aggregation Metadata is to enable discoverability of the content aggregation and to provide descriptive information about the Content Aggregation as a whole.
- **Content Organization Metadata:** Content Organization Metadata describes the content organization. The purpose of applying Content Organization Metadata is to enable discoverability within, for example, a content repository and to provide descriptive information about the content structure, as a whole, defined by the content organization.
- **Activity Metadata:** Activity Metadata describes an individual activity. The purpose of applying Activity Metadata is to make the activity accessible (enabling discovery) within a content repository. The metadata should describe the activity as a whole.
- **SCO Metadata:** Metadata can be applied to SCOs to provide descriptive information about the content in the SCO independent of use. This metadata is used to facilitate reuse and discoverability of content.
- **Asset Metadata:** Metadata can be applied to assets to provide descriptive information about the assets independent of any usage or potential usage within courseware content. This metadata is used to facilitate reuse and discoverability, within, for example, a content repository during content creation.

The LOM data model is a hierarchy of elements, as it is illustrated in Figure 3-3.

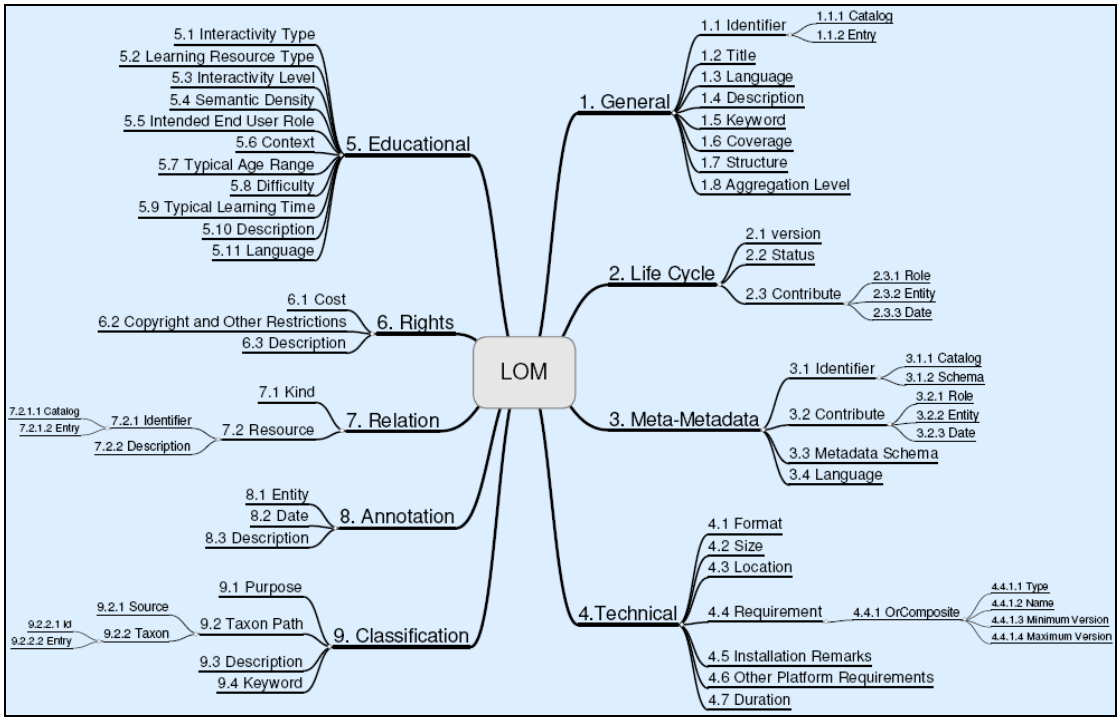


Figure 3-3: A schematic representation of the hierarchy of elements in the LOM data model [30**]**

At the topmost level there are nine general categories, each of which comprises several elements. These elements may be leafs that hold data or may themselves be aggregate elements and so on. The semantics of an element are determined by its context: they are affected by the parent or container element in the hierarchy and by other elements in the same container. For each element, the LOM data model defines the name of the element by which it is referenced, the number of values allowed (size) and whether the order of the values is significant (order). In addition, it contains an explanation for each data element and an illustrative example. Finally, LOM defines a datatype for each leaf element, as well as a value space that specifies the encoding of the data for that element. The general categories of the LOM data model can be used as follows to describe the SCORM Content Model Components [2]:

1. The *General* category can be used to describe general information about the SCORM Content Model Component as a whole.
2. The *Life Cycle* category can be used to describe features related to the history and current state of the SCORM Content Model Component and those who have affected the component during its evolution.
3. The *Meta-metadata* category can be used to describe information about the metadata record itself (rather than the SCORM Content Model Component that the record describes).

4. The *Technical* category can be used to describe technical requirements and characteristics of the SCORM Content Model Components.
5. The *Educational* category can be used to describe the educational and pedagogic characteristics of the SCORM Content Model Component.
6. The *Rights* category can be used to describe the intellectual property rights and conditions of use for the SCORM Content Model Component.
7. The *Relation* category can be used to describe features that define the relationship between this SCORM Content Model Component and other targeted components.
8. The *Annotation* category can be used to provide comments on the educational use of the SCORM Content Model Component and information on when and by whom the comments were created.
9. The *Classification* category can be used to describe where the SCORM Content Model Component falls within a particular classification system.

Each LOM metadata element is optional; therefore the content authors can choose which elements to use. However, content authors must be extremely careful when adding metadata to their content because the amount and type of metadata greatly affects the probability of their content to be discovered and reused.

3.1.4 Sequencing and Presentation

Content sequencing is part of the learning experience, and as such, it must be included in the content package. The Content Aggregation Model defines how the sequencing behaviors described in SCORM SN Book can be encoded in XML and placed in the manifest file. There are two ways to embed sequencing information in the manifest file: either by using the <sequencing> element which encapsulates all of the necessary sequencing information for a given activity, or by using the <sequencingCollection> element which acts as a container for sequencing information that can be reused by several activities(Listing 3-5).

```
<item identifier = "activity_2" identifierref = "SEQ01" parameters = "?tc=CM-
01&amp;act=2">
    <title>Activity 2</title>
    <imsss:sequencing IDRef="seqInfo">
        <imsss:objectives>
            <imsss:primaryObjective satisfiedByMeasure="true">
                 <imsss:minNormalizedMeasure>0.8</imsss:minNormalizedMeasure>
             </imsss:primaryObjective>
        </imsss:objectives>
    </imsss:sequencing>
</item>
```

```
<imsss:sequencingCollection>
  <imsss:sequencing ID="seqInfo">
      <imsss:rollupRules >
         <imsss:rollupRule childActivitySet = "all">
            <imsss:rollupConditions>
               <imsss:rollupCondition operator = "not" condition = "attempted"/>
            </imsss:rollupConditions>
            <imsss:rollupAction action = "satisfied"/>
         </imsss:rollupRule>
      </imsss:rollupRules>
      </imsss:sequencing>
   </imsss:sequencingCollection>
```

Listing 3-5: Example of a <sequencing> element

Sequencing is associated with the activity and the organization components of the Content Model. In the context of IMS Content Packages, this is captured by including the <sequencing> element within either an <item> element or an <organization> element; the <organization> element is very much similar to the <item> element except that it is the root element of the hierarchical structure. If a <sequencing> element uses both the IDRef attribute and inline definition of sequencing information, any top-level element defined inline overrides any similar element defined in the referenced element. Adding sequencing information to the content package is optional; SCORM defines a default sequencing behavior based on which the learner can choose activities at will.

SCORM also defines presentation information for each <item>. More specifically it defines which of the navigations controls (i.e. previous, next) will be hidden from the learner (Listing 3-6) during his interaction with the corresponding content object.

```
<item identifier = "activity 2" identifierref = "SEQ01" parameters = "?tc=CM-
01&amp;act=2">
    <title>Activity 2</title>
    <adlnav:presentation>
         <adlnav:navigationInterface>
            <adlnav:hideLMSUI>continue</adlnav:hideLMSUI>
            <adlnav:hideLMSUI>previous</adlnav:hideLMSUI>
            <adlnav:hideLMSUI>suspendAll</adlnav:hideLMSUI>
        </adlnav:navigationInterface>
    </adlnav:presentation>
</item>
```

Listing 3-6: Example of a <adlnav:presentation> element

3.2 Run-Time Environment

The SCORM Run-Time Environment (RTE) book describes how a learning management system (LMS) launches content packages and how those packages communicate with the LMS in a consistent manner. More specifically it describes a data model that holds information relevant to the learner's experience with the content, a Run-Time API for the communication between the LMS and content packages as well as a common management mechanism to support this communication.

3.2.1 Run-Time Environment (RTE) Management

In response to some navigation request, the LMS is responsible to determine which learning activity to deliver next and then launch the corresponding content object. The selection of the appropriate learning activity to be launched is based on the interactions between learner and content objects and the sequencing rules defined in the manifest file. At this point, it is useful to introduce some terms as defined from the *Institute of Electrical and Electronics Engineers (IEEE) 1484.11.1 Standard for Learning Technology – Data Model for Content Object Communication*:

- **Learner Attempt:** A tracked effort by a learner to satisfy the requirements of a learning activity that uses a content object. An attempt may span one or more learner sessions and may be suspended between learner sessions.
- **Learner Session:** An uninterrupted period of time during which a learner is accessing a content object.
- **Communication Session:** An active connection between a content object and an application programming interface.
- **Login Session**: A period of time during which a learner begins a session with a system (logged on) until the time the learner terminates the session with the system (logged out).

In the context of SCORM, a "Learner Attempt" begins as soon as a learning activity is identified for delivery. When a new learner attempt begins, the LMS is required to create and initialize a new set of run-time data for the SCO to access and use (run-time data are discussed in the next section). During the attempt, the learner will be engaged with a content object (either a SCO or Asset), at which point the "Learner Session" begins. If the content object is a SCO, it must establish a "Communication Session" with the runtime

environment once it has been launched. Technically, the SCO has to locate the run-time API instance and call the Initialize() function (the run-time API is discussed in the next section). Once the session has been successfully initialized, the SCO can exchange data with the LMS through the corresponding functions of the API instance. Typically, the communication session ends due to a learner- or system-triggered navigation event; the SCO terminates the communication with the LMS by calling the Terminate() function of the API instance and the content object is taken away. At this point, the learner session ends, leaving the SCO either in a normal state or in a suspended state. In the latter case the "Learner Attempt" is not yet over. The learner can later resume the attempt and continue his effort to complete the SCO. When the attempt is resumed, the LMS must ensure that any data set prior to the suspension are available to the SCO.

3.2.2 Application Programming Interface (API)

The API described in the current version of SCORM is based on *IEEE 1484.11.2 – Standard for ECMAScript Application Program Interface (API) for Content to Runtime Service Communication* which has its origins to the *AICC's CMI001 Guidelines for Interoperability*. The use of a common API provides a standardized way for SCOs to communicate with LMSs, fulfilling this way many of SCORM's high-level requirements for interoperability and reuse. In simple terms, the API is a collection of predefined functions that govern the communication between the SCO and the LMS. These functions can be divided into the following categories:

- **Session functions** that are used to initiate and terminate the "Communication Session" between the SCO and the LMS.
- **Data-transfer functions** are used to exchange data between the SCO and the LMS.
- **Support functions** are used for auxiliary communications (e.g., error handling) between the SCO and the LMS.

At this point, it is important to clarify that the communication established between the SCO and the LMS is unilateral, from the SCO to the LMS. There are no functions defined at the SCO's side that can be called from the LMS. Even in the case of the API Instance returning a value, this is done in response to a call initiated by the SCO.

The session functions defined by SCORM's API are the Initialize() and Terminate() functions. The Initialize() function is used to initiate the communication session and allows to handle LMS specific initialization issues. All SCOs must call the Initialize() function prior to any

further communication. It requires a single parameter, an empty character string, and returns a character string that can be either equal to “true” or “false”. A return value equal to “true” denotes that the “Communication Session” was successfully initialized. If the “Communication Session” was not successfully initialized, “false” is returned and the SCO can seek more information about the error encountered by calling the GetLastError()and GetDiagnostic() functions. These functions can be called an arbitrary number of times during the “Communication Session” and provide information about the last error encountered. The Terminate() function is used to terminate the “Communication Session”. It requires a single parameter, an empty character string; if the “Communication Session” was successfully terminated it returns a character string equal to “true”, otherwise it returns “false”. The Terminate() function also causes the persistence of any data set by the SCO by calling the Commit() function.

The Data-transfer functions include the GetValue(), the SetValue and the Commit() functions. The GetValue() function is used to request data from the LMS. It takes a single parameter as input; specifically a character string that specifies which data model element’s value will be retrieved. If no error occurs, the value is returned, otherwise an empty character string is returned. However, if an empty string is returned, SCO cannot safely assume that an error occurred because the value of an element can be an empty string. The SetValue function() allows the SCO to send data to the LMS for storage. It requires two character strings as input parameters. The data model element denoted by the first parameter will be set to the value contained in the second parameter. The function returns a character string equal to “true” if no errors occurs, otherwise it returns “false”. For example, an error may occur if the data model element specified from the first parameter does not exist. Finally, the Commit() function requests from the runtime environment to forward the current data of the API instance to the persistent data store. Although the Commit() function will be called implicitly when the “Communication Session” ends, a SCO should call it explicitly at regular intervals during the “Communication Session” to minimize data loss due to unexpected events such as a browser crash.

The support functions which are included in the API are the GetLastError(), the GetErrorString() and the GetDiagnostic() functions. The GetLastError() function requires an empty character string as input and returns an error code that represents the last error encountered during SCO’s interaction with the RTE API instance. SCORM requires an error state to be maintained for every API Instance. Error state is set to 0 when a function (except

the support functions) executes successfully, otherwise it is set to the error code. Thus, this function actually returns the error state set by the last function call. The GetErrorString() requires a single input parameter, a character string containing an error code, and returns a human readable description of that error code. For example, calling GetErrorString() with "405" as the input parameter, will return a string containing "Data Model Element Is Write Only" indicating that a SCO attempted to retrieve a data model value for an element that is implemented as write-only. The GetDiagnostic() function provides additional diagnostic information for the API Instance. It requires a character string as input parameter which can be, but not limited to, an error code.

3.2.3 SCORM Run-Time Environment Data Model

The SCORM Run-Time Environment data model is based on the *IEEE standard 1484.11.1: Data Model for Content to Learning Management System Communication*. This standard has evolved from the data model defined in *the AICC Computer Managed Instruction (CMI) Guidelines for Interoperability* which was used in SCORM 1.2. For this reason the SCORM Run-Time Environment data model is usually referred to as CMI data model. SCORM RTE data model provides a well-defined set of data model elements used to hold the information being tracked by a SCO. Since the IEEE standard defines only the data type and the value space for each data element it is left to SCORM to define their behavior and relationship with the rest of the reference model.

Because the IEEE communication data model standard is abstract, ADL has developed a dot notation binding for it. A similar dot notation binding was also used for the navigation data model introduced by ADL. The first is identified in the SCORM dot notation by the "cmi.", while the second with "the adl.nav.". Each data model element can hold either one data record per learner session or an array of data records; a data record represents a collection of data of related elements. The data record is accessed using an index value representing its position in the array.

```
GetValue("cmi.learner_name")
SetValue("cmi.objectives.0.completion_status")
```

Listing 3-7: SCORM's Dot Notation

The data model elements are used to track information like status, scores, interactions, objectives, etc. The value of some data model elements can impact the value of others

or/and the sequencing of the learning activities. The following table contains high level information regarding the elements included in the RTE data model.

Data Model Element	Dot-Notation Binding	Description
Comments From Learner	cmi.comments_from_learner	Contains feedback collected from the user during the learning experience.
Comments From LMS	cmi.comments_from_lms	Contains comments and annotations intended to be made available to all learners (e.g. hints during an evaluation activity).
Completion Status	cmi.completion_status	Indicates whether the learner has completed the SCO (i.e. completed, incomplete, not attempted, and unknown).
Completion Threshold	cmi.completion_threshold	Identifies a threshold after which the SCO is considered completed.
Credit	cmi.credit	Indicates whether the learner will be credited for performance in this SCO.
Entry	cmi.entry	Indicates how a SCO will be initialized according to whether the learner has previously accessed the SCO and how a previous attempt on the SCO ended.
Exit	cmi.exit	Indicates how or why the learner left the SCO.
Interactions	cmi.interactions	Contains information pertaining to an interaction for the purpose of measurement or assessment.
Launch Data	cmi.launch_data	Provides data specific to a SCO that the SCO can use for initialization.
Learner Id	cmi.learner_id	Identifies the learner experiencing the SCO.
Learner Name	cmi.learner_name	Contains the name of learner experiencing the SCO.
Learner Preference	cmi.learner_preference	Contains learner preferences associated with the learner's use of the SCO.
Location	cmi.location	Specifies a location in the SCO (e.g. bookmark, checkpoint).
Maximum Time Allowed	cmi.max_time_allowed	Indicates the amount of accumulated time the learner is allowed to use a SCO in the learner attempt.

Mode	cmi.mode	Identifies the modes in which the SCO may be presented to the learner (i.e. browse, normal, and review).
Objectives	cmi.objectives	Specifies learning or performance in terms of objectives associated with a SCO.
Progress Measure	cmi.progress_measure	Identifies a measure of the progress the learner has made toward completing the SCO.
Scaled Passing Score	cmi.scaled_passing_score	Identifies a threshold after which the SCO is considered mastered.
Score	cmi.score	Identifies the learner's score for the SCO.
Session Time	cmi.session_time	Identifies the amount of time that the learner has spent in the current learner session for the SCO.
Success Status	cmi.success_status	Indicates whether the learner has mastered the SCO.
Suspend Data	cmi.suspend_data	Provides information created by a SCO during a learning experience. This information may be used by the SCO to resume a suspended learner attempt.
Time Limit Action	cmi.time_limit_action	Indicates what the SCO should do when the maximum time allowed is exceeded.
Total Time	cmi.total_time	Identifies the sum of all of the learner's learner session times accumulated in the current learner attempt prior to the current learner session.

Table 3-1: SCORM Run-Time Environment Data Model Elements

3.3 Sequencing and Navigation

SCORM 2004 enables content designers to represent the intended behavior of an authored learning experience such that any SCORM compliant LMS can consistently realize it by properly sequencing discrete learning activities. To provide this capability SCORM incorporates the *IMS Simple Sequencing (IMS SS) specification* The *IMS Simple Sequencing* specification is labeled as simple because it includes a limited number of widely used sequencing behaviors, not because the specification itself is simple. Actually, the IMS SS specification has received a lot of criticism in the e-learning community as extremely

complicated and difficult to implement both for content developers and LMS vendors. The SCORM SN book defines how IMS SS is applied and extended in a SCORM environment and which mechanisms a LMS must possesses to properly process sequencing information at run-time. Finally, SCORM SN defines a Navigation data model that allows a SCO to communicate to the run-time environment a navigation request. The Navigation data model also allows SCOs to enable/disable navigation controls at will and request information about their validity.

3.3.1 Activity Tree and Clusters

SCORM SN uses a content structure diagram, called the "Activity Tree", to describe the hierarchical structure of learning activities. Using this diagram, SCORM SN book describes informational and processing requirements such as sequencing algorithms and behaviors regardless of the underlying implementation. Figure 3-4 presents an example of an activity tree.

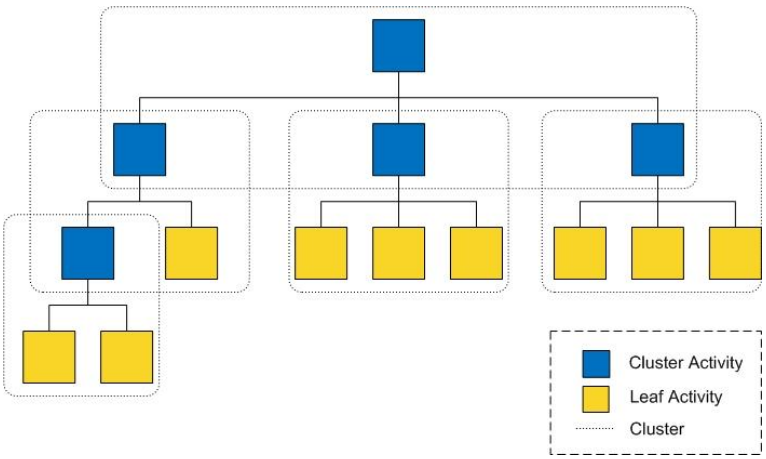


Figure 3-4: Activity Tree

As we have already mentioned in Section 3.2.2, the SCORM CAM provides a hierarchical structure, represented by the <organization> element of the manifest file, for the organization of learning content. Thus, an "Activity Tree" can be easily derived from the manifest file by mapping the <organization> element to the root of the Activity Tree and each <item> element to a learning activity. However, the "Activity Tree" is an abstract concept that represents an instance of hierarchical learning activities and the corresponding sequencing information attached to each learning activity. SCORM does not pose any requirements on how an "Activity Tree" is implemented or when it is instantiated. Moreover, SCORM does not require an "Activity Tree" to be static during the learning

experience. Its structure can be dynamically altered as long as the final outcome is in accordance to the one defined by IMS SS.

Another important concept within IMS SS is the “cluster”. The cluster is a basic building block of the Activity Tree that includes a single parent activity (usually referred to as cluster activity) and its immediate children, but not the descendants of its children. Leaf activities are not considered as clusters. Clusters are playing an important role in IMS SS because each sequencing strategy is defined in the context of a cluster on which the parent activity contains all the sequencing information.

3.3.2 Tracking Model

SCORM 2004 has introduced an additional data model, namely the *Tracking Model*, to support its sequencing capabilities. The *Tracking Model* is a set of data model elements used to hold the current sequencing state of each activity in the Activity Tree and the Activity Tree as a whole. All activities have associated tracking status information, per each learner, which is dynamically updated to reflect the learner interactions with the corresponding SCOs. When a learning activity is experienced for first time all of the *Tracking Model* elements are initialized to default pre-defined values. Many elements of the SCORM Run-Time data model directly correspond to elements of the *Tracking Model*. Thus, SCORM provides bi-directional mapping directions between the Run-Time data model and the *Tracking Data Model* to ensure that changes in one are properly propagated to the other. In this way, both data models are accurately reflect the learner interactions.

The *Tracking Data Model*, unlike the Run-Time Data Model, is associated with all activities in the Activity Tree. Thus, it naturally comes into question how the *Tracking Data Model* is updated in the case of cluster activities where no runtime information is available. SCORM SN defines how *Tracking Model* elements are updated through a process called rollup behavior. The rollup behavior specifies how the tracking status of a cluster activity is updated according to the current tracking status of its children. The *Tracking Model* consists of four sets of tracking status information which are discussed in the following subsections.

Objective Progress Information

Objective Progress Information describes the learner’s progress during a learning experience in terms of the pre-defined objectives in the manifest file or/and the objectives defined at

runtime by the SCO. For each attempt on an activity, a learner gets one set of objective progress information for each objective. The following table describes what is included in the *Objective Progress Information*.

Name	Description	Value Space	Default Value
Objective Progress Status	Indicates the objective has a satisfaction value (True or False).	Boolean	False
Objective Satisfied Status	Indicates the objective is satisfied (True or False). The determination or meaning of satisfied or not satisfied is not defined in this model. The value is unreliable unless *Objective Progress Status* is True.	Boolean	False
Objective Measure Status	Indicates the objective has a measure value (True or False).	Boolean	False
Objective Normalized Measure	The measure (e.g. score) for the objective, normalized between -1..1 (inclusive). The mechanism to normalize a measure is not defined in this model. The value is unreliable unless *Objective Measure Status* is True.	Real [-1..1] Precision of at least 4 significant decimal digits	0.0

Table 3-2: Objective Progress Information Elements

Activity Progress Information

Activity Progress Information describes a learner's overall progress on an individual activity. The corresponding information is tracked only if the activity has defined as tracked in manifest file and it spans all attempts on that activity. The following table outlines the content of *Activity Progress Information*.

Name	Description	Value Space	Default Value
Activity Progress Status	Indicates the activity progress information is (True or False) meaningful for the activity.	Boolean	False
Activity Absolute Duration	The cumulative duration of all attempts on the activity, i.e., the time from the initial start of the activity to the end of the activity. The mechanism for determining the	Duration Accuracy 0.1	0.0

	duration is not defined in this model. The value is unreliable unless *Activity Progress Status* is True.	second	
Activity Experienced Duration	The cumulative *experienced* duration of all attempts on the activity, i.e., the time from the initial start of the activity to the end of the activity, not including any time elapsed while the activity is suspended (i.e., when the activity is not being experienced or is inactive). The mechanism for determining the duration of the suspend time is not defined in this model. The value is unreliable unless *Activity Progress Status* is True and the activity is a leaf.	Duration Accuracy 0.1 second	0.0
Activity Attempt Count	The number of attempts on the activity. The count includes the current attempt, i.e., 0 means the activity was not attempted and 1 or greater means it either is in progress or completed. The value is unreliable unless *Activity Progress Status* is True.	Non Negative Integer	0

Table 3-3: Activity Progress Information Elements

Attempt Progress Information

Attempt Progress Information describes a learner's progress in the context of a unique attempt on a tracked activity. This information is instantiated for each new attempt on an activity. The following table shows what is included in *Attempt Progress Information*.

Name	Description	Value Space	Default Value
Attempt Progress Status	Indicates the attempt progress information (True or False) is meaningful for the activity attempt. The value is unreliable unless *Activity Attempt Count* is greater than (>) 0.	Boolean	False
Attempt Completion Amount	The measure of the completion of the attempt on the activity, normalized between 0..1 (inclusive) where 1 means the activity attempt is complete and any lesser value means the activity attempt is not complete. The mechanism to define the completion amount is not defined in this model. The value is unreliable unless *Attempt Progress Status* is True.	Real [0..1] Precision of at least 4 significant decimal digits	0.0

Attempt Completion Status	Indicates the activity attempt is completed (True or False). The determination or meaning of completed or incomplete is not defined in this model. The value is unreliable unless *Attempt Progress Status* is True.	Boolean	False
Attempt Absolute Duration	The duration of the attempt on the activity, i.e., time from the start of the attempt to the end of the attempt. The mechanism for determining the duration is not defined in this model. The value is unreliable unless *Attempt Progress Status* is True.	Duration Accuracy 0.1 second	0.0
Attempt Experienced Duration	The *experienced* duration of the attempt on the activity, i.e., the time from the start of the attempt to the end of the attempt, not including elapsed time while the activity attempt is suspended (i.e., when the activity attempt is not being experienced or is inactive). The mechanism for determining the duration or the suspend time is not defined in this model. The value is unreliable unless *Attempt Progress Status* is True.	Duration Accuracy 0.1 second	0.0

Table 3-4: Attempt Progress Information Elements

Activity State Information

Activity State Information describes the status of an activity per learner and contains the following:

Name	Description	Value Space	Default Value
Activity is Active	Indicates that an attempt is currently in progress for the activity, i.e. the activity has been delivered to the learner and has not been terminated, or the activity is an ancestor of the *Current Activity (True or False).*	Boolean	False
Activity is Suspended	Indicates the activity is currently suspended (True or False).	Boolean	False
Available Children	A list indicating the ordering of the available child activities for the activity.	Ordered List of Activities	All children

Table 3-5: Activity State Information Elements

Global State Information

Global State Information describes the status of the Activity Tree per learner. This information is initiated once for an Activity Tree for each learner. The following table contains the various types of *Global State Information*.

Name	Description	Value Space	Default Value
Current Activity	Indicates either the activity that is being experienced by the learner or the last terminated activity.	Activity	None
Suspended Activity	Indicates the activity from which a *Suspend All* navigation request was triggered.	Activity	None

Table 3-6: Global State Information Elements

3.3.3 Sequencing Definition Model

The *SCORM Sequencing Definition Model* defines a set of elements that may be used by content developers to describe a learning experience in terms of the sequence in which the learning content is presented to the learner. The sequencing definition model elements are applied to learning activities within the context of an Activity Tree and apply to all learners accessing the activity tree.

3.3.3.1 Sequencing Control Modes

The *Sequencing Control Mode* elements determine the different types of navigation that are available to the learner, thus they can be used to constrain the learning experience. Moreover, the *Use Current Attempt Objective Information* and *Use Current Attempt Progress Information* elements determine what tracking information will be used during rollup and rule evaluations. A content developer can use none or multiple control modes on an activity according to his needs. If a control mode element is not explicitly set the default value must be used.

Name	Description	Value Space	Default Value
Sequencing Control Choice	Indicates that a *Choice* sequencing request is permitted (True or False) to target the children of the activity.	Boolean	True
Sequencing	Indicates that an active child of this activity is	Boolean	True

Control Choice Exit	permitted to terminate (True or False) if a *Choice* sequencing request is processed.		
Sequencing Control Flow	Indicates whether linear navigation (continue, previous) is permitted.	Boolean	False
Sequencing Control Forward Only	Indicates that backward targets (in terms of Activity Tree traversal) are not permitted (True or False) for the children of this activity.	Boolean	False
Use Current Attempt Objective Information	Indicates that the objective progress information for the children of the activity will only be used (True or False) in rule evaluations and rollup if that information was recorded during the current attempt on the activity.	Boolean	True
Use Current Attempt Progress Information	Indicates that the attempt progress information for the children of the activity will only be used (True or False) in rule evaluations and rollup if that information was recorded during the current attempt on the activity.	Boolean	True

Table 3-7: Sequencing Control Mode Elements

The Sequencing *Control Choice*, *Sequencing Control Flow* and *Sequencing Control Forward Only* modes affect only the children of the activity that are applied, thus it's pointless to define them in leaf activities.

3.3.3.2 Limit Conditions

A content developer can define one or more limit conditions to constrain the access to an activity. The IMS SS states that conditions can be based on time of day, time spent on the activity and number of attempts. However, SCORM does not require the evaluation of any time-based limit conditions. The only way of implementing a limit condition in the context of SCORM is based on the number of attempts on an activity. For example a content developer may specify that an evaluation activity cannot be attempted more than once.

3.3.3.3 Sequencing Rules

Sequencing rule elements represent a rule-based sequencing model where each sequencing rule consists of a set of conditions and a corresponding action. Zero or more rules can be applied to activities which are evaluated against the current tracking information. If the

conditions are met the specified action is applied. Figure 3-5 illustrates the structure of a sequencing rule.

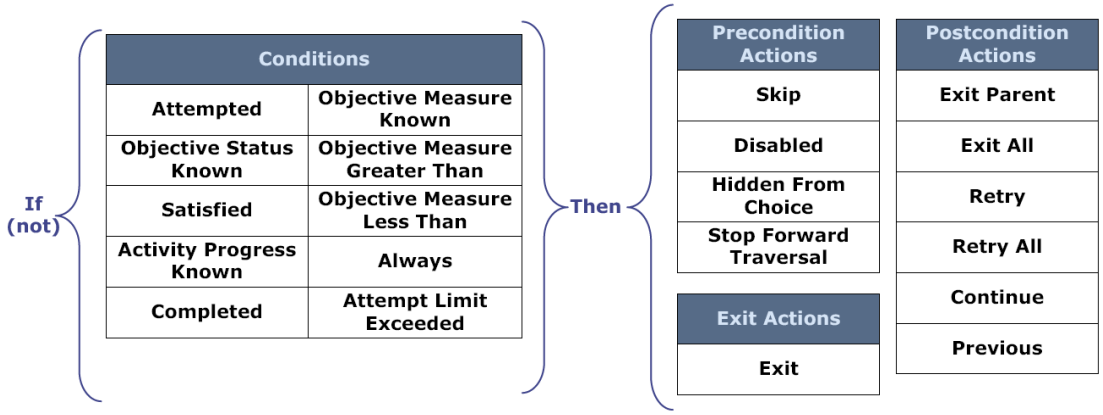


Figure 3-5: Structure of Sequencing Rules

Sequencing rule conditions can be combined under the control of either an "*any*" (logical Or) operator or an "*all*" (logical And) operator. In the first case, the condition set evaluates to true if any of the individual conditions evaluates to true while in the second the condition set evaluates to true if all of the individual rule conditions evaluate to true. Each rule condition evaluation can be negated using the logical operator not. According to when the rule is evaluated the set of actions can be categorized as follows:

- **Precondition Actions** are applied when traversing the Activity Tree to identify an activity for delivery and control whether an activity will be identified for delivery or not.
    - **Skip** action causes the activity to be skipped during the Activity Tree traversal.
    - **Disabled** action prevents the activity from being delivered in any way.
    - **Hidden From Choice** prevents the activity from being delivered due to a choice request. Moreover the activity will not be visible in the table of contents.
    - **Stop Forward Traversal** action prevents the user from proceeding past this activity and it applies only in the case of a choice request.
- **Post Condition Actions** are applied when an attempt on an activity terminates.
    - **Exit Parent** action terminates the attempt on the activity's parent.
    - **Exit All** action terminates all attempts on activities in the activity tree.
    - **Retry** action causes the activity to be delivered again to the learner.
    - **Retry All** action causes the Activity Tree to be experienced again.

- **Continue** action delivers the next available activity to the learner.
- **Previous** action delivers the previous available activity to the learner.

- **Exit Actions** are applied after the termination of a descendant activity's attempt.
  - **Exit** action results the unconditional termination of the activity.

3.3.3.4 Sequencing Rollup Rules

As we have briefly discussed in 3.4.3 cluster activities act as containers of activities and they are not associated with content objects; an activity cannot include both content objects and content packages. Thus, there is no direct way for their tracking information to be updated (i.e. by applying the mapping rules defined in SCORM from the *Run-Time Data Model* to the *Tracking Data Model*). IMS SS defines how tracking information of child activities affects the tracking information of their parent. An arbitrarily number of Rollup Rules can be applied to a cluster activity where each rule consists of a set of child activities to be included in the evaluation process, a set of conditions evaluated against the current tracking information of the included child activities and an action that updates the cluster activity's tracking information if the conditions are met. It is obvious that Rollup Rules will have no effect if defined on leaf activities.

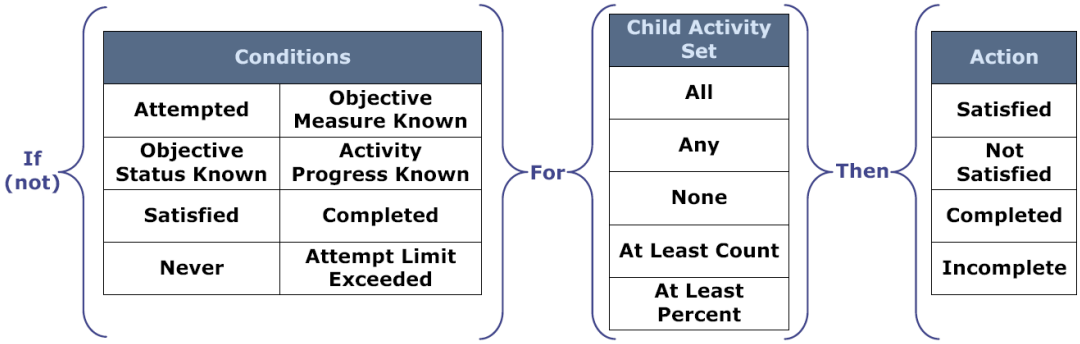


Figure 3-6: Structure of Rollup Rules

Rollup rule conditions can be combined under the control of either an "*any*" (logical Or) operator or an "*all*" (logical And) operator. In the first case, the condition set evaluates to true if any of the individual conditions evaluates to true while in the second the condition set evaluates to true if all of the individual rule conditions evaluate to true. Each rule condition evaluation can be negated using the logical operator not. The rollup child activity set element determines how each condition will be evaluated. The following tokens are applicable:

- **All**: The specified rollup action is applied if the condition combination evaluates to True for all included child activities**.** This is the default token used if the child activity set element is not explicitly specified.
- **Any**: The specified rollup action is applied if the condition combination evaluates to True for any of the included child activities.
- **None**: The specified rollup action is applied if the condition combination evaluates to True for none of the included child activities.
- **At Least Count**: The specified rollup action is applied if the condition combination evaluates to True for at least a threshold count of the included child activities.
- **At Least Percent**: The specified rollup action is applied if the condition combination evaluates to True for at least a threshold percentage of the included child activities.

By default, the condition combination is evaluated against the tracking information of all child activities of a cluster. However, the IMS SS specification provides the content developer with the capability to specify child activities or parts of tracking information of child activities that will not be used during rollup conditions evaluations ADL, as a response to the demands of the e-learning community, added more options for further refinement of the tracking information used in rollup.

Finally, if no rollup rules are defined on a cluster activity that have the actions satisfied or not satisfied the default rollup rules will be applied by the sequencing engine. The default rollup rules are:

- If the objective status of all immediate children is known, then not satisfied
- If all immediate children are satisfied, then satisfied

Similarly, if no rollup rules are defined on a cluster activity that have the actions completed or incomplete the default rollup rules will be applied by the sequencing engine. The default rollup rules are:

- If the activity progress of all immediate children is known, then incomplete
- If all immediate children are completed, then completed

3.3.3.5 Learning Objectives

SCORM 2004 allows content developers to define learning objectives and associate them with learning activities. SCORM does not pose any requirements on the semantics of a learning objective nor how a learning objective is interpreted. It is a general purpose

element and its simple design allows for many uses. It can be used to capture what knowledge, skills and abilities learners acquired during a learning experience or fairly to track if a learner mastered a SCO. Learning objectives are often used to control sequencing behaviors. Each activity must have one and only one primary objective and may have an arbitrarily number of objectives. The primary objective is pretty much similar to the conventional objectives except that it contributes to rollup (rollup is discussed in next sections). Each learning objective is defined using the elements shown in the following table.

Name	Description	Value Space	Default Value
Objective ID	The identifier of an objective associated with the activity. The ID is a link to the corresponding objective information.	Unique Identifier	None Required Value
Objective Satisfied by Measure	Indicates that the *Objective Minimum Satisfied Normalized Measure* is to be used (True or False) in place of any other method to determine if the objective associated with the activity has been satisfied.	Boolean	False
Objective Minimum Satisfied Normalized Measure	The minimum satisfaction measure for the objective, normalized between -1..1 (inclusive). If the *Objective Measure Status* for the objective is True and the *Objective Normalized Measure* for the objective exceeds this value, the *Objective Progress Status* is set to True and the *Objective Satisfied Status* is set to True. If the *Objective Measure Status* for the objective is True and the *Objective Normalized Measure* for the objective is less than this value, the *Objective Progress Status* is set to True and the *Objective Satisfied Status* is set to False. The value is unreliable unless *Objective Satisfied by Measure* is True.	Real [-1..1] Precision of at least 4 significant decimal digits	1.0
Objective Contributes to Rollup	Indicates that the *Objective Satisfied Status* and *Objective Normalized Measure* for the objective are used (True or False) during rollup.	Boolean	False

Table 3-8: Learning Objective Elements

By default, the objective status information associated with an activity's objective is local to that activity (local objectives). Other activities cannot directly access the progress

information associated with another activity's objectives. However, SCORM allows defining objectives that are globally scoped (shared global objectives) which can be referenced by local objectives. Multiple objectives, associated with the same or different activities may reference the same global shared objective, thus sharing its objective status information. This enables a content developer, for example, to define a sequencing strategy where an activity becomes available to the learner only if he successfully completes a precedent evaluation activity. How a local objective is related with a shared global is specified by one or multiple Objective Maps. An Objective Map defines a mapping of an activity's local tracking information to and from a shared global objective and it is applied whenever the status information associated with a local objective is updated or retrieved. Objective progress information retrieval from a shared global objective does not alter the state of the local tracking data. The IMS SS specification defines the following elements for the description of an Objective Map.

Name	Description	Value Space	Default Value
Activity Objective ID	The identifier of the local objective associated with the activity.	Unique Identifier	None Value is Required
Target Objective ID	The identifier of global shared objective targeted for the mapping.	Unique Identifier	None Value is Required
Read Objective Satisfied Status	Indicates that the *Objective Satisfied Status* for the identified local objective (*Activity Objective ID*), should be retrieved (True or False) from the identified shared global objective (*Target Objective ID*), when the progress for the local objective is undefined - *Objective Progress Status* for the identified local objective (*Activity Objective ID*) is False. This operation does not change the Objective Information associated with the local objective.	Boolean	True
Write Objective Satisfied Status	Indicates that the *Objective Progress Status* and *Objective Satisfied Status* values, for the identified local objective (*Activity Objective ID*), should be transferred (True or False) to the identified global shared objective (*Target Objective ID*), upon termination of an attempt on the activity.	Boolean	False

Read Objective Normalized Measure	Indicates that the *Objective Normalized Measure* for the identified local objective (*Activity Objective ID*), should be retrieved (True or False) from the identified shared global objective (*Target Objective ID*), when the measure for the local objective is undefined - *Objective Measure Status* for the identified local objective (*Activity Objective ID*) is False. This operation does not change the Objective Information associated with the local objective.	Boolean	True
Write Objective Normalized Measure	Indicates that the *Objective Measure Status* and *Objective Normalized Measure* values, for the identified local objective (*Activity Objective ID*), should be transferred (True or False) to the identified global shared objective (*Target Objective ID*), upon termination of an attempt on the activity.	Boolean	False

Table 3-9: Objective Map Elements

3.3.4 Navigation Model

Almost every LMS provides some sort of navigation mechanism that learners will use to navigate across the learning content. SCORM 1.2 didn't pose any requirements on what navigation controls an LMS must offer, nor provided content objects with the capability to interact with the provided navigation controls. ADL realized these shortcomings and added a well defined navigation model in SCORM 2004 that also supports the incorporated IMS SS specification.

A SCORM 2004 compliant LMS must provide a set of user interface devices that corresponds to the different navigation requests defined in SCORM's navigation model. Typically, when a learner triggers such a device, the LMS is responsible to translate the event into its corresponding navigation request, process the navigation request through its sequencing implementation and eventually, deliver the appropriate activity to the learner or end the communication session. The LMS is also responsible for ensuring that the offered user interface devices correspond to valid navigation requests; their processing will result in either the identification of a deliverable content object or the end of the current communication session without an exception being raised. For example, if a learner currently experiences the first available activity of an activity tree, a navigation device that corresponds to a *Previous* navigation request must be disabled because there is no previous activity to be delivered. Moreover, SCORM 2004 allows a SCO to get information concerning

the availability of the LMS's navigation controls, or even issue navigation requests directly to the LMS.

It is important to clarify that the SCORM SN book does not impose any requirements on the type or look and feel of the user interface presented to a learner during the learner experience. A LMS developer is free to provide a user interface based on individual needs as long as a learner has a way of triggering the appropriate navigation events defined in the navigation data model.

3.3.4.1 Navigation events

As mentioned before events can be triggered either by a learner through the offered user interface devices or by SCOs using the navigation data model that we discuss later on. Nevertheless they will be processed by the sequencing implementation in the exact same manner. In the case that a learner triggers a navigation event while a navigation request has already been issued by the SCO, the navigation event triggered by the user will take precedence; the navigation request issued by the SCO will be discarded. The following table lists the navigation events defined in SCORM 2004 and describes the expected behavior for each one. It also indicates the mapping between navigation requests and navigation events and the possible source for each navigation request.

Navigation Event	Source	Behavior Description
Start	LMS	This navigation event requests from the LMS to deliver the first available activity in the activity tree. In most cases this event will be automatically triggered by the LMS when the learner selects an Activity Tree to interact with. This navigation event results in a Start Navigation Request.
Resume All	LMS	This navigation event requests from the LMS to resume a previously suspended attempt on the root activity of the activity tree. In most cases this event will be automatically triggered by the LMS when the learner selects a previously suspended Activity Tree to interact with. This navigation event results in a Resume All Navigation Request.
Continue	LMS or SCO	This navigation event requests from the LMS to deliver the next logical activity available in the activity tree. This event results in a Continue Navigation Request. In addition an Exit Termination Request is issued if the current activity is still active.
Previous	LMS or	This navigation event requests from the LMS to deliver the previous

	SCO	logical activity available in the activity tree. This event results in a Previous navigation request. In addition an Exit Termination Request is issued if the current activity is still active.
Choose	LMS or SCO	This navigation event requests from the LMS to deliver a specific activity in the activity tree. This event results in a Choice Navigation Request. In addition an Exit Termination Request is issued if the current activity is still active.
Jump	LMS or SCO	This navigation event requests from the LMS to deliver a specific activity in the activity tree. The Jump Navigation Event is new to SCORM 2004 4th Edition. It is similar to the Choice Navigation Event apart from the fact that it results to the direct delivery of the specified activity and no sequencing rule evaluations take place. This event results in a Jump Navigation Request. In addition an Exit Termination Request is issued if the current activity is still active.
Abandon	LMS or SCO	This navigation event requests from the LMS to prematurely or abnormally terminate the current attempt on the current activity. The attempt cannot later be resumed. This event results in an Abandon Navigation Request.
Abandon All	LMS or SCO	This navigation event requests from the LMS to prematurely or abnormally terminate the current attempt on the root activity of the activity tree. The attempt cannot be later resumed. This event results in an Abandon All Navigation Request.
Suspend All	LMS or SCO	This navigation event requests from the LMS to suspend the current attempt on the root activity of the activity tree. The attempt doesn't end and can be later resumed. Tracking information that already has been recorded will be available when the attempt resumes. This event results in a Suspend All Navigation Request.
Unqualified Exit	LMS or SCO	This navigation event requests from the LMS to end the current attempt on the current activity. This event results in an Exit navigation request.
Exit All	LMS or SCO	This navigation event requests from the LMS to end the current attempt on the root activity of the activity tree. This event results in an Exit All navigation request.

Table 3-10: SCORM Navigation Events

If an attempt on the root activity of the Activity Tree suspends or ends the same applies to all active learning activities. When a navigation event is triggered from whatever source the LMS will forward the corresponding navigation request to the sequencing engine. The

sequencing engine will process the navigation request resulting to one of the following states:

- A learning activity is identified for delivery and the LMS launches the corresponding content object.
- No learning activity is identified for delivery. In this case, SCORM does not pose any requirements on the subsequent LMS behavior but recommends providing some sort of guidance to the user and avoiding leaving him at a dead-end situation.
- An exception occurs and no learning activity is identified for delivery. In this case, SCORM does not pose any requirements on the subsequent LMS behavior but recommends to gracefully handle the exception.
- The communication session ends and control is returned to the LMS (i.e. due to an Exit All navigation request).

3.3.4.2 Navigation Data Model

SCORM allows a SCO to navigate the learner across the Activity Tree by directly issuing navigation requests. This navigation request will be processed by the LMS upon termination, unless a learner triggers a navigation event before the SCO terminates. A navigation request is indicated by the SCO by using the SetValue() function of the Run-Time Environment API to set the adl.nav.request element of the Navigation Data Model . This can be done multiple times during the communication session, with every call overwriting the current value of the adl.nav.request. Thus, upon SCO's termination the last navigation request will be honored.

A SCO is permitted to contain user interface devices that trigger navigations events as the LMS's user interface devices do. However, unlike the LMS, a SCO does not know if a navigation request is valid at runtime in order to control in an appropriate manner what user interfaces will offer to the learner. The navigation data model of SCORM contains this information which is retrievable by a SCO using the GetValue() function of the Run-Time Environment API. Thus, a SCO can adjust accordingly the user interface that it will provide to the learner.

Name	Description	Value Space
adl.nav.request	This element indicates the navigation request the SCO would like to be processed by the LMS after its termination. .	"_none_" (default value) "continue" "previous" "choice {target}" "jump {target}" "abandon" "abandonAll" "exit" "exitAll" "suspendAll"
adl.nav.valid_request.continue	This element holds information about the validity of the Continue Navigation Request.	"true" "false" "unknown"(default value)
adl.nav.valid_request.previous	This element holds information about the validity of the Previous Navigation Request.	"true" "false" "unknown"(default value)
adl.nav.valid_request.choice{target}	This element holds information about the validity of the Choice Navigation Request on the learning activity indicated by the target.	"true" "false" "unknown"(default value)
adl.nav.valid_request.jump{target}	This element holds information about the validity of the Jump Navigation Request on the learning activity indicated by the target.	"true" "false" "unknown"(default value)

Table 3-11: Navigation Data Model Elements

Chapter 4: **Implementing SCORM**

This chapter describes the design and implementation of an architecture for integrating the latest version of the Sharable Content Object Reference Model (SCORM 2004 4th Edition) into a pre-existent learning management system. The proposed architecture deals with the following issues:

- The importing of any SCORM 2004 conformant content package.
- The launching of content objects.
- The runtime communication between content objects and the LMS.
- The sequencing of learning activities according to the rules defined in the content package.
- The tracking of learner progress.
- The provision of an intelligent user interface.

The proposed architecture was implemented in the *eFront* learning management system which is used in the course *CS-100 Introduction to Computer Science* offered by the Computer Science Department of the University of Crete. The implementation was successfully validated using the *SCORM 2004 4th Edition Version 1.1.1 Test Suite* provided by the ADL.

4.1 Architecture Overview

The proposed architecture presumes a pre-existent learning management system that uses a relational database as its persistent data store. The LMS must provide a user and course management system for enrolling learners in courses and a Web-based course delivery system. The proposed architecture follows a client-server approach as illustrated in Figure 4-1.

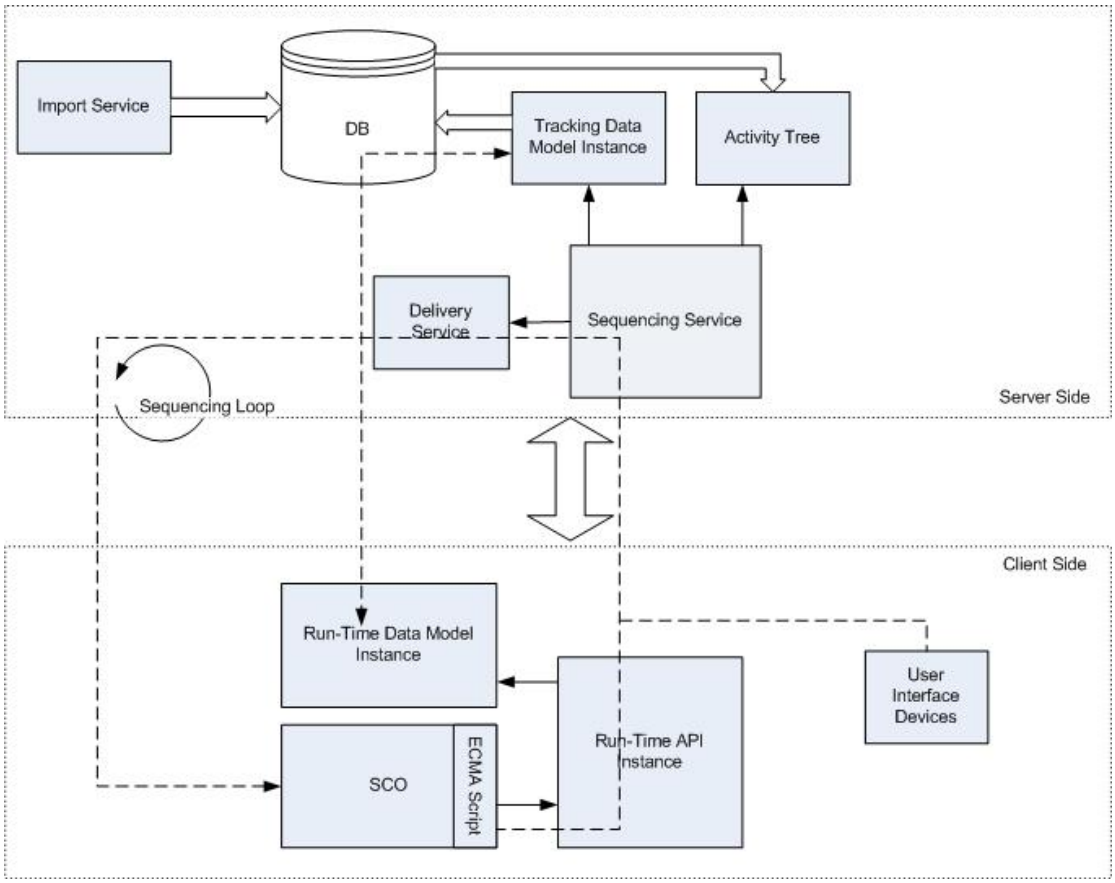


Figure 4-1: The proposed architecture

The client side comprises the *Run-Time API Instance* and the *Run-Time Data Model Instance*, both implemented in JavaScript. The delivery service is responsible for delivering content objects associated with learning activities. A content object communicates, via JavaScript, with the Runtime API Instance for exchanging data with the *Run-Time Data Model Instance* and managing the communication session.

The server side component is implemented in PHP and consists of the *Import Service*, the *Sequencing Service*, the *Tracking Data Model Instance*, the *Activity Tree* and the relational data store. The *Import Service* is responsible for importing SCORM content

packages into the LMS. The *Activity Tree* provides a run-time treelike representation of the hierarchical structure defined in the <organization> element of the manifest file and expose a set of methods for accessing it. The *Tracking Data Model Instance* captures information about the learner's interaction with content objects during a learning experience. The *Sequencing Service* traverses the Activity Tree based on a sequencing request, applying sequencing rules to locate the appropriate activity to deliver to the learner. Each sequencing rule is evaluated against the *Tracking Data Model Instance*.

The client side component communicates navigation requests to the Sequencing Service via HTTP Requests. The *Sequencing Service* identifies the next Activity to be delivered to the learner and ultimately the associated content object is delivered to the learner through LMS's *Delivery Service*. Whenever the content object calls the Commit() function of the Run-Time API, the *Run-Time Data Model Instance* is forwarded to the persistent data store via an AJAX[15] call.

[15] AJAX (Asynchronous JavaScript with XML) allows sending HTTP or HTTPS requests directly to a web server and getting the server response data directly to the browser.

4.2 Content Package Importing

SCORM compliant learning content is wrapped into self-contained packages that include the actual content, as well as a manifest file (imsmanifest.xml) describing the content structure and the associated resources of the package. Each content package is stored in an archive format, most often in the format of PKZip which SCORM recommends. The import process begins with the unpacking of the archive file. The actual content is stored in the file system while the manifest is forwarded for further processing.

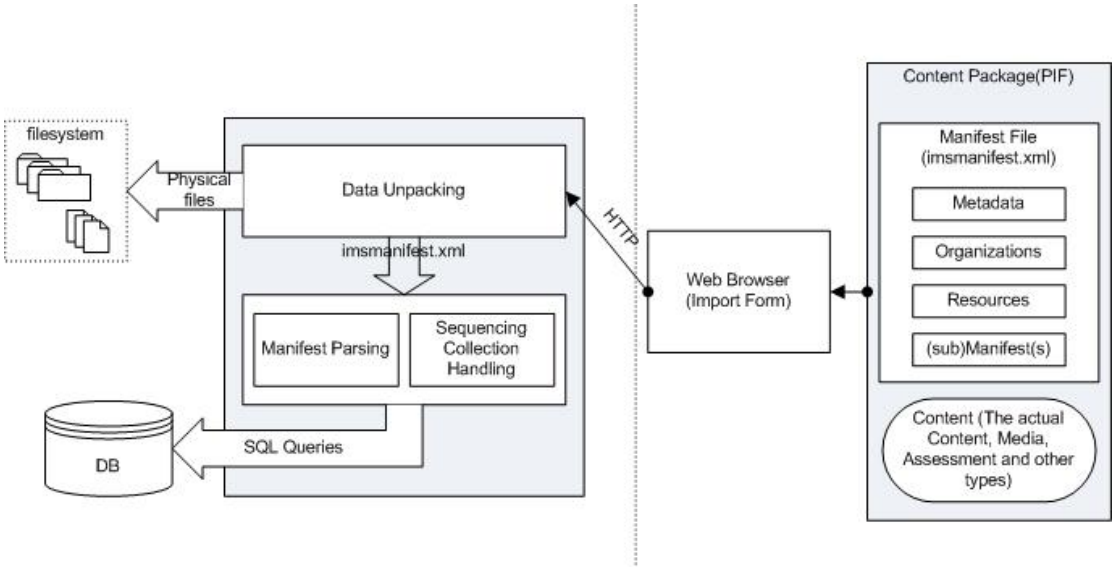


Figure 4-2: The Import Process

The manifest file is parsed to an array of arrays where each sub-array represents an element of the manifest file. It is important to note that the hierarchical and sibling order of the manifest file must be preserved in order to accurately derive the Activity Tree later on. Therefore, each sub-array contains two elements holding the element's parent and children. Next, the possible sequencing collections residing into the array are handled. The <sequencingCollection> element contains sequencing information to be referenced and reused by multiple nodes in the content structure. This information is attached to the proper nodes according to the following rules specified by SCORM:

- The <sequencingCollection> collection is "merged" with any sequencing information applied directly to the learning activity.
- "Top-level" sequencing elements applied directly to a learning activity (an <item> or <organization>) take precedence over identical elements defined in the referenced sequencing collection. If a "top-level" XML element occurs both in the node and the

referenced sequencing collection, the referenced element, along with its descendants, is ignored.

Afterwards, the array is traversed and a series of SQL statements handle its insertion to the database. The complete database schema can be found in the appendix. The centric table of the database is the *Content* table, as it holds the hierarchical and sibling order of the activities. Each record corresponds to an activity where the *parent_content_ID* element holds the activity's parent and the *previous_content_ID* the previous activity in a preorder traversal. Moreover, since the SCORM tracking and sequencing model is activity-oriented, the rest of the tables, containing either sequencing or tracking information, reference the *Content* table. Finally, the *data* element contains information about the resource associated with the activity. More specifically, it includes the html code for an iframe that loads the corresponding resource (either SCO or asset) from the file system. The relative path to the resource is constructed according to the values of the *xml:base* and the *href* attributes of the resource element and the parameters attribute of the item element. The xml:base attribute specifies a path offset for the resource, as defined in the XML Base specification [31]. The *href* is a Uniform Resource Locator (URL) representing the "launching point" of the resource. Finally, the parameters attribute contains a set of parameters used internally by the resource.

4.3 Implementing the Run-Time Environment

As we have already discussed in 3.3, the SCORM RTE book specifies the LMS responsibilities concerning the launch of the content packages, the communication between the LMS and the content packages and the tracking of the learner's interactions with the content package. In this section, we illustrate how to ground the SCORM Run-Time Environment to a real learning management system. We will focus on the launching of content objects, the runtime API and data model implementation, as well as the navigation interface provided by the LMS.

4.3.1 Launching Content Objects

When a learning activity is identified for delivery, the LMS is responsible for launching the content object associated with that activity. Figure 4-3 depicts a launched content object associated with *Activity 1*.

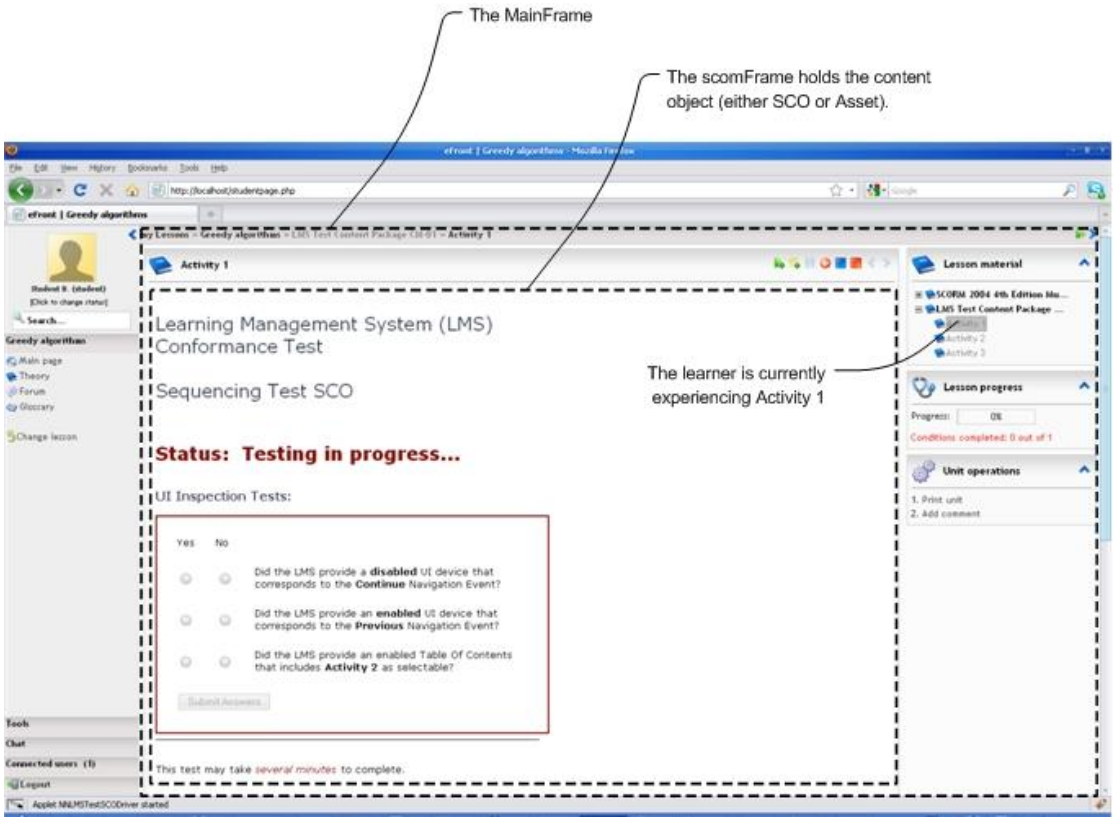


Figure 4-3: Example of a Launched Content Object

The content object is launched inside the *Main Frame*. More specifically, the *Main Frame* loads the html code contained in the *data* element of the *Content* table that corresponds to the activity identified for delivery. In this example, the html code loaded by the *Main Frame* is:

```
<iframe height = "100%"  width = "100%" frameborder = "no" name = "scormFrame" id =
"scormFrame" src = "content/lessons/5/LMSTestPackage_CM-
01/resources/SequencingTest.htm?tc=CM-01&act=1"></iframe>
```

Listing 4-1: Example of an iframe loading a Content Object

Let's focus on the *src* property of the iframe element that indicates the location of the resource. The *"content/lessons/5/LMSTestPackage_CM-01/"* represents the location to which the actual content of the content package were placed. The "resources/" and the "SequencingTest.htm" are derived correspondingly from the *xml:base* and the *href* attributes of the resource element while the "?tc=CM-01&act=3" part is derived from the *parameter* attribute of the item element. It is worth mentioning that the *href* attribute is also permitted to contain any external fully qualified URL.

```
<item identifier = "activity_1" identifierref = "SEQ01" parameters = "?tc=CM-
01&amp;act=1">
    <title>Activity 1</title>
    . . . . .
</item>
```

```
<resource identifier="SEQ01"
          type="webcontent" adlcp:scormType="sco"
          href="SequencingTest.htm"
          xml:base="resources/">
    . . . . .
</resource>
```

Listing 4-2: Parts of the manifest file specifying the location of a content object and how it should be launched

4.3.2 Run-Time Data Model

The SCORM RTE data model provides a well-defined set of data model elements that are used to hold the information being tracked by a SCO. A SCO must utilize only these predefined data model elements using the Run-Time API's GetValue() and SetValue() functions in order to ensure its reusability. The run-time data model is implemented in JavaScript as a hierarchy of objects that reflects the hierarchy of the CMI data model. This enables the direct access of each data model element according to the dot notation used by the API's data transfer functions. A special case is the data model elements that collect sets of data (records). These elements are represented as array objects where each record of the array holds an object representing a set of data.

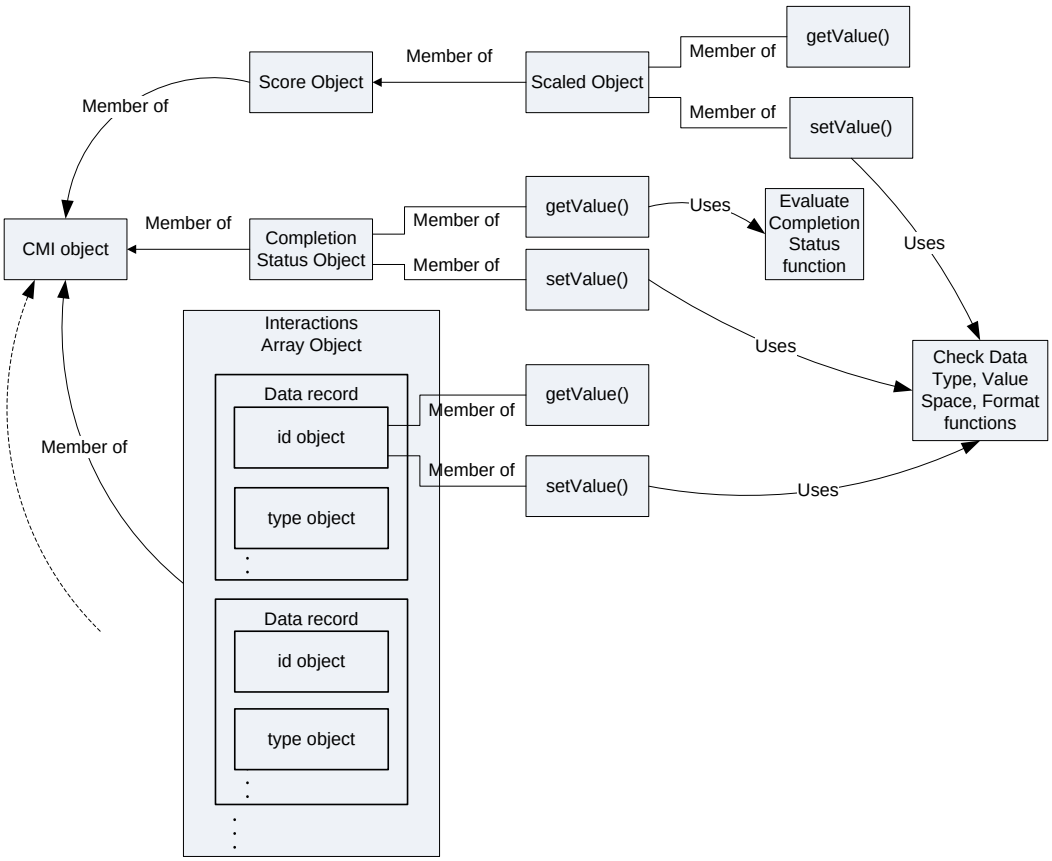


Figure 4-4: CMI Data Model as a hierarchy of JS Objects

Each object representing a leaf element in the data model hierarchy has a property named "value" that holds the actual value of the element and getter and setter methods for accessing the value property. SCORM requires some of the data model elements to be implemented as "write only" or "read only". If the SCO attempts to retrieve a data model value for a "write only" element, the getter method will set the error status of the API instance to the appropriate error code. The same applies for the setter method when a SCO attempts to set a data model value for a "read only" element.

Each data model element has a data type, a value space and a format that places additional restrictions to the element's value. For example, time may have a certain format (e.g. hh:mm:ss). All parameters are passed to the API functions as character strings, therefore a set of auxiliary functions has been implemented that check if the value to be set to a specific element is compatible with the data type and the format described by the data model and if it is in the allowed value space. These functions are relying on regular expressions[16] and they are called by the elements' setter functions as needed.

Moreover, SCORM defines how the value of some data model elements is correlated to the value of others. The value of Completion Status is affected by the values of Completion Threshold and Progress Measure while the value of Success Status is affected by the values of Scaled Score and Scaled Passing Score. Two auxiliary functions have been implemented, namely the evaluateCompletionStatus() and the evaluateSuccessStatus(), that receive the values of the correlated data model elements and perform the appropriate transformation. Thus, when the value of either the Completion Status or the Success Status element is requested (via a GetValue() call), the appropriate auxiliary function is called and the transformed value is returned.

Persist Storage

SCORM does not pose any requirements to the LMS on the persistence or access to previous learner attempt data. However, it requires the persistence of data in attempts that span across multiple learner sessions; a learner attempt may be suspended, hence remaining incomplete and can be resumed in a later learner session. Therefore, we implemented a persistence mechanism on top of a relational database.

When a suspended attempt is resumed the following data model elements must be initialized according to the previous learner attempt data:

- Location (cmi.location)
- Suspend Data (cmi.suspend_data)
- Learner Preference (cmi.learner_preference)
- Interactions (cmi.interactions)
- Comments from LMS (cmi.comments_from_lms)
- Comments from Learner (cmi.comments_from_learner)

[16] Regular expressions (also referred to as regex or regexp) are a powerful and standardized way of searching, replacing, and parsing strings using complex patterns of characters.

- Progress Measure (cmi.progress_measure)
- Completion Status (cmi.completion_status)
- Score (cmi.score)
- Success Status (cmi.success_status)
- Objectives (cmi.objectives)

The Commit() function collects the values of those elements from the data model instance and passes them to a server-side PHP script via an AJAX call. The PHP script checks if the currently experienced content object ends due to an *Abandon* or *Abandon All* navigation request. If this is the case the script terminates, as an abandoned learner attempt must not modify the state of the SCO's associated activity. Otherwise, it stores the following elements directly to the database:

- Location (cmi.location)
- Suspend Data (cmi.suspend_data)
- Learner Preference (cmi.learner_preference)
- Interactions (cmi.interactions)
- Comments from LMS (cmi.comments_from_lms)
- Comments from Learner (cmi.comments_from_learner)

As previously discussed, SCORM 2004 has introduced an additional data model, namely the IMS SS *Tracking Data Model*, which is used to hold dynamic sequencing state information associated with each activity in the Activity Tree for each learner. This information must persist at least until the current attempt on the root activity of the Activity Tree ends; this information, as later discussed, is also stored in the database. Parts of the Run-Time Data Model and the IMS SS *Tracking Data Model* are closely related. SCORM defines how the following SCO Run-Time Data Model elements are mapped to IMS SS *Tracking Data Model* elements and vice versa:

- Progress Measure (cmi.progress_measure)
- Completion Status (cmi.completion_status)
- Score (cmi.score)
- Success Status (cmi.success_status)
- Objectives (cmi.objectives)

Therefore, instead of managing overlapping information stored into different locations we chose a different approach. The PHP script updates the IMS SS *Tracking Data Model Instance* according to the mappings defined and discards the values of the corresponding Run-Time Data Model elements. When the values of the Run-Time Data Model elements are requested from the SCO, the corresponding Sequencing *Tracking Data Model* elements are

retrieved and their values are properly translated according to the mapping defined before returned.

Data Model Initialization

When a SCO is launched the Run-Time Data Model is initialized using values from the LMS and the manifest file; after the import process the information that the manifest file carries resides in the database. The elements Learner Name (cmi.learner_name) and Learner ID (cmi.learner_id) are initialized from the LMS's database according to which learner launched the SCO. The elements Completion Threshold (cmi.completion_threshold), Launch Data (cmi.launch_data), Maximum Time Allowed (cmi.max_time_allowed), Scaled Passing Score (cmi_scaled_passing_score) and Time Limit Action (cmi.time_limit_action) are initialized from the manifest. In addition, as SCORM requires, the Objectives collection (cmi.objectives) is populated with the objectives defined in the manifest file (children of the <imsss:objectives> element). Finally, if the SCO launched was previously attempted and the attempt was suspended, then the previous Run-Time Environment data model element values are used to initialize the new Data Model instance. This happens even in the case where the activity associated with the launched SCO is defined as not tracked.

4.3.3 Run-Time Application Programming Interface (API)

When the launch process is completed, if the launched content object is a SCO it will immediately seek the API Instance to begin tracking the learner's experience with the LMS. According to the previously discussed "Web Assumption", content objects are launched in a Web browser. SCORM requires from the SCO to seek the API Instance in the following locations of the Web browser's DOM, in the order specified:

1. The chain of parents of the current window, if any exist, until the top of the window of the parent chain is reached
2. The opener window, if any
3. The chain of parents of the opener window, if any exist, until the top window of the parent chain is reached

Since we attempt to accomplish a transparent integration of SCORM to the LMS, as far as the learner is concerned, we use the current window to load the content object. The Run-Time API, along with the Data Model Instance, is placed in the immediate parent frame of the iframe that loads the content object, as shown in Figure 4-5.

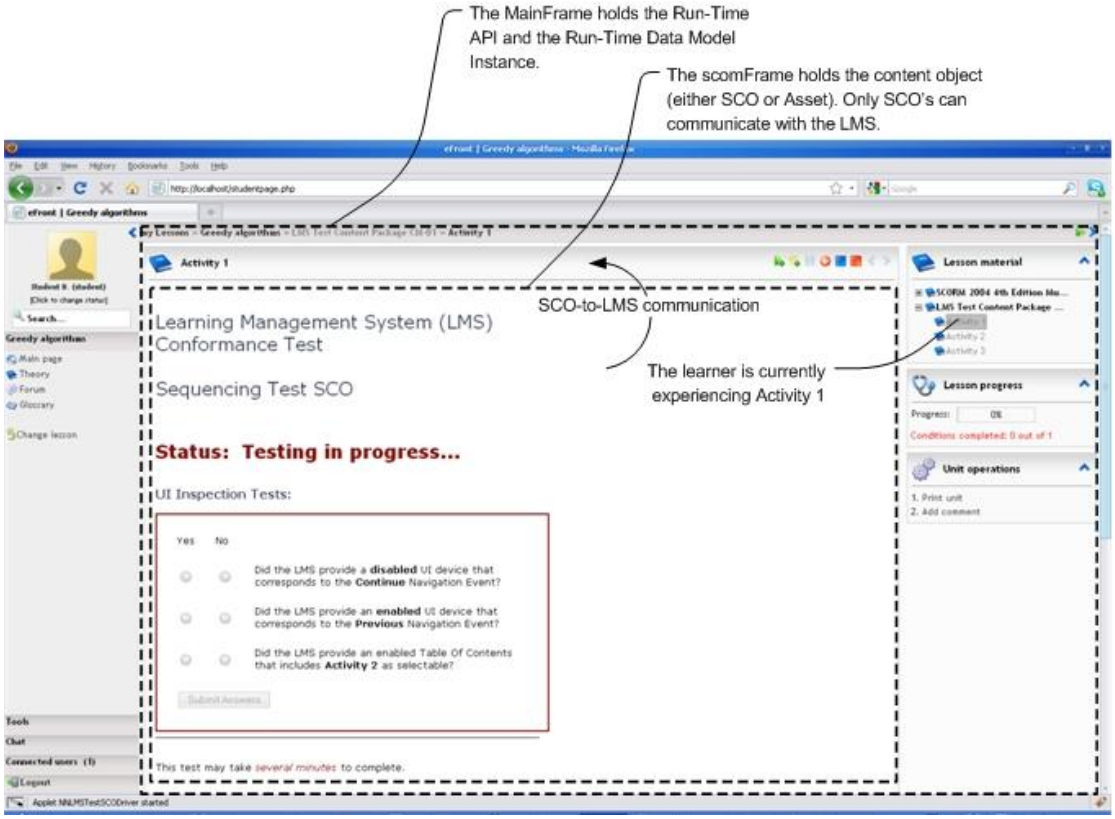


Figure 4-5: SCO-to-LMS Communication

As is required by SCORM, the Run-Time API has been implemented as a DOM object, named API_1484_11, that exposes a set of methods defined in the ECMAScript language, more

commonly known as JavaScript. Here, we briefly describe how these functions were implemented. Conceptually each function adheres to the following pattern:

1. Check if the function is allowed under the current state of the "Communication Session".
2. Check the input parameters.
3. Perform the intended action of the function.
4. Set the API's error status to "0" if no errors encountered, otherwise to the error code (this step is omitted in the case of support functions).

The current state of the *Communication Session* is kept in a global variable, namely the *CurrentState* variable. According to the state of the *Communication Session*, different functions are allowed. The following table shows the possible states of the *Communication Session* and the allowed functions for each state.

States	Allowed Functions
Not Initialized (default)	Initialize(), GetLastError(), GetErrorString(), GetDiagnostic()
Running	GetValue(), SetValue(), Commit(), Terminate(), GetLastError(), GetErrorString(), GetDiagnostic()
Terminated	GetLastError(), GetErrorString(), GetDiagnostic()

Table 4-1: Allowed functions for each state of the *Communication Session*

Session functions

The Initialize() function assigns to a global variable, named *CurrentState*, the value *Running* that indicates that the *Communication Session* has started. Similarly, the Terminate() function assigns to the *CurrentState* the value *Terminated* that indicates that the Communication Session has terminated. The Terminate() function also makes an implicit call to the Commit() function.

Data Transfer functions

The GetValue() function seeks an element in the data model instance and returns its value. It uses an auxiliary function, the checkParameter() function, which checks whether the data model element holds a single or multiple data records. In the first case, it retrieves and returns the value of the single data record, while in the second case it traverses the data record array to find and then retrieve and return the requested record. This function also checks if the requested element exists in the data model instance. If this isn't the case, it

sets the corresponding error code to the error status of the API Instance. The SetValue() function seeks an element in the data model instance and sets its value. This function also uses the checkParameter() function to determine whether the data model element is meant to hold a single or multiple data records. In the former case, it simply sets its value while in the latter case, it first finds the next index position in the array and then sets the value of the specified data model element. This is necessary because the index position is significant to the SCO as it uses it to get or re-set particular data records of a data model element. The Commit() function forwards the current data of the Runt-Time Data Model Instance to the persistent data store as previously discussed.

Support functions

The API object has three properties supporting the error handling. The *errorCode* property of the API object holds the error status of the API instance. It is updated whenever a Data Transfer or Session function is called. If no error occurs, it is set to “0”, otherwise it is set to the error code. The *errorString* property is an associative array where each key is an error code and each value a textual description for the error code. Similarly, the *errorDiagnostic* property is an associative array where each key is an error code and each value a string with diagnostic information about the error code. The GetLastError() function returns the value of the *errorCode* property. The GetErrorString() and GetDiagnositc() functions receive an error code as input and return the corresponding value from the *errorString* and *errorDiagnostic* arrays.

4.4 Sequencing and Navigation

SCORM 2004, through the *IMS Simple Sequencing* specification, enables content designers to represent the intended behavior of an authored learning experience such that any SCORM compliant LMS can consistently realize it by properly sequencing discrete learning activities. In order to be SCORM compliant, a LMS must implement the sequencing part of the reference model. The same does not apply for learning content as it can be SCORM compliant even if it does not include any sequencing information. This section describes our approach in implementing the sequencing behaviors defined in SCORM SN Book and integrating them in a real learning management system.

4.4.1 Deriving the Activity Tree

The Activity Tree is a fundamental structure in the SCORM SN book, since the included sequencing behaviors are described in terms of traversing the nodes of the tree. The Activity Tree is the run-time representation of the hierarchical structure defined in the <organization> element of the manifest file. This structure essentially provides a means of organizing learning content. The <organization> element is the root of the Activity Tree and each of its <item> elements corresponds to a learning activity. As discussed in 1.1 following the import process, the hierarchical structure of the <organization> element is preserved in the "Content" table of the database. The learning management system *eFront*, on which the proposed architecture was implemented, has an internal proprietary representation for hierarchically structured content, namely the *ContentTree* class. The *ContentTree* class defines a hierarchical tree-like content representation using nested arrays consisting of elements such as name, timestamp etc to represent discrete learning units. The same approach is used for the SCORM activity tree, by replacing the learning unit with a SCORM activity. In addition, the sequencing information of each activity, defined by the Sequencing Definition Model, is attached to a series of pre-defined elements within the "activity" array.

The SCORM sequencing behaviors defined in the *IMS Simple Sequencing* specification perform certain actions on the Activity Tree. To that end, the *ContentTreeSCORM* class was defined which extends the *ContentTree* class and implements the following methods.

Name	Description
getNodeChildren(node)	Given a node as input, this function returns its children nodes.
getNodeAncsestors(node)	Given a node as input, this function returns its ancestor nodes up to the root node (including the current node).
getRootNode()	This function returns the root node of the Activity Tree.
getLastNode()	This function returns the last node of the Activity Tree.
isLeaf(node)	Given a node as input, this function returns true if it does not contain any children nodes, otherwise it returns false(cluster node).
getNextNode(node)	Given a node as input, this function returns the next available node in a preorder traversal[17].
getPreviousNode(node)	Given a node as input, this function returns the next available node in a reverse preorder traversal.
getNextSiblingNode(node)	Given a node as input, this function returns the next available sibling node in a preorder traversal.
getPreviousSiblingNode(node)	Given a node as input, this function returns the next available sibling node in a reverse preorder traversal.
findCommonAncestor(node, node)	This function takes two nodes as input and returns their common ancestor node.
formNodePath(node, node)	This function takes two nodes as input and returns the path from the first to the second in the form of an array of nodes (inclusive)

Table 4-2: Member functions of the *ContentTree* class

Typically, the *ContentTreeSCORM* class instantiates whenever a Navigation Request is triggered.

4.4.2 Tracking Data Model

The *Tracking Data Model* captures information about the learner's interaction with the content objects associated with activities during a learning experience. The values of the *Tracking Data Model* elements are used by the sequencing engine to evaluate the conditional rules defined by the *Sequencing Definition Model*. SCORM does not impose any implementation requirements on how the *Tracking Model* is represented or managed. In our proposed architecture the *Tracking Data Model* is directly mapped to tables in the relational

[17] The IMS SS specification defines the default traversal of the activity tree as pre-order traversal.

database. Whenever a Commit() request is processed by the SCO the corresponding tables are accordingly updated via an AJAX call to reflect the state of the Run-Time Data Model, as described in 1.2.2. The *Tracking Data Model* information is made available to the sequencing process through a series of objects, members of the *ContentTreeSCORM* class, that instantiate when the sequencing process starts. Each object is initialized from the corresponding tables of the relational data store.

Activity Progress Information

Activity Progress Information describes a learner's cumulative progress information across all attempts on an activity in the activity tree. It is managed through the *activityProgressInformation* object which contains the *Activity Progress Information* for each activity in the Activity Tree for the current learner. The *activityProgressInformation* object provides the following methods.

Methods	Description
update(activity, values)	Updates the Activity Progress Information associated with the *activity* according to the *values* parameter.
get(activity)	Returns the Activity Progress Information associated with the *node*.
increaseAttempts(activity)	Increases the number of attempts on the activity.
commit()	Forwards the Activity Progress Information to the relational data store.

Table 4-3: Members of the *ActivityProgressInformation* class

The activity parameter is a unique identifier assigned to each activity during the import process. The values parameter is an associative array with keys that corresponds to the *Activity Progress Information* elements*: Activity Progress Status*, *Activity Absolute Duration*, *Activity Experienced Duration* and *Activity Attempt Count*.

Activity State Information

Activity State Information indicates the state (i.e. active, suspended, and undefined) of an activity per learner. It is managed through the *activityStateInformation* object which contains the Activity State Information for each activity in the Activity Tree for the current learner. The *activityStateInformation* object provides the following methods.

Methods	Description
update(activity, values)	Updates the Activity Progress Information associated with the *activity* according to the *values* parameter.
isActive(activity)	Checks if the *active* is active.
isSuspended(activity)	Checks if the *node* is in suspended.
commit()	Forwards the Activity State Information to the relational data store.

Table 4-4: Member functions of the *ActivityStateInformation* class

The *activity* parameter is a unique identifier assigned to each activity during the import process. The values parameter is an associative array with keys that corresponds to the *Activity State Information* elements: *Activity is Active* and *Activity is Suspended*.

Global State Information

Global State Information describes the state of the Activity Tree per learner. It is managed through the *globalStateInformation* object which contains the Global State Information of the Activity Tree for the current learner. The *globalStateInformation* object provides the following methods.

Methods	Description
update(values)	Updates the Global State Information of the Activity Tree according to the values parameter.
get()	Returns the Global State Information of the Activity Tree.
commit()	Forwards the Global State Information to the relational data store.

Table 4-5: Member functions of the *GlobalStateInformation* class

The values parameter is an associative array with keys that corresponds to the *Global State Information* elements: *Current Activity* and *Suspended Activity*.

Attempt Progress Information

Attempt Progress Information describes a learner's progress on an activity during a single attempt that may span across multiple learner sessions. It is managed through the *attemptProgressInformation* object which contains the *Attempt Progress Information* for each Activity in the Activity Tree for the current learner.

Methods	Description
update(activity, values)	Updates the Attempt Progress Information associated with the activity according to the values parameter.
get(activity)	Returns the Attempt Progress Information associated with the activity.
initialize(activity)	Initializes the Attempt Progress Information associated with the activity to the default values defined by SCORM.
commit()	Forwards the Attempt Progress Information to the relational data store.

Table 4-6: Member functions of the *AttemptProgressInformation* class

The activity parameter is a unique identifier assigned to each activity during the import process. The values parameter is an associative array with keys that corresponds to the *Attempt Progress Information* elements: *Attempt Progress Status*, *Attempt Completion Amount*, *Attempt Completion Status*, *Attempt Absolute Duration* and *Attempt Experienced Duration*.

Objective Progress Information

Objective Progress Information describes the learner's progress during a learning experience in terms of the pre-defined objectives in the manifest file or/and the objectives defined at runtime by the SCO. *Objective Progress Information* maintained for local objectives is managed through the objectives object. It contains the *Objective Progress Information* associated with each objective in the Activity Tree and provides the following methods.

Methods	Description
getObjectives(activity)	Returns the objectives collection associated with the activity.
insertObjective(activity, values)	Inserts a new objective to the objectives collection associated with the activity.
inManifest(activity, objective)	Checks if the objective is defined by the manifest.
getObjectiveInfo(activity, objective, flag)	Returns the Objective Progress Information associated with the objective. The flag determines if read objective maps will be applied.

updateObjectiveInfo(activity, objective, values)	Updates the Objective Progress Information associated with the objective according to the values parameter. If any write objectives maps exist they are applied.
initialize(activity, objective)	Initializes the Objective Progress Information associated with the objective to the default values defined by SCORM.
commit()	Forwards the Objective Progress Information to the relational data store.

Table 4-7: Member functions of the *Objectives* class

The *activity* parameter is a unique identifier assigned to each activity during the import process. The objective parameter is the identifier of a local objective as defined in the manifest file. The values parameter is an associative array with keys that corresponds to the *Objective Progress Information* elements: *Objective Progress Status*, *Objective Satisfied Status*, *Objective Measure Status* and *Objective Normalized Measure*.

Objective Progress Information maintained for shared objectives is managed through the *sharedObjectives* object. Because shared objectives may be shared across multiple activity trees within the LMS the *sharedObjectives* object contains the shared objectives of all the activity trees stored into the LMS. It provides the following methods.

Methods	Description
updateObjectiveInfo(objective, values)	Updates the Objective Progress Information associated with the shared objective according to the values parameter.
getObjectiveInfo(objective)	Returns the Objective Progress Information associated with the shared objective.
initialize(objective)	Initializes the Objective Progress Information associated with the shared objective to the default values defined by SCORM.
commit()	Forwards the Objective Progress Information to the relational data store.

Table 4-8: Member functions of the *SharedObjectives* class

The *objective* parameter is the identifier of a shared objective as defined in the manifest file. The values parameter is an associative array with keys that corresponds to the *Objective Progress Information* elements: *Objective Progress Status*, *Objective Satisfied Status*,

Objective Measure Status and *Objective Normalized Measure*. The methods provided by the *SharedObjectives* object are used by the *Objectives* object to apply any *Objective Maps* defined.

4.4.3 Overall Sequencing Process

Within the SCORM context, the learning content is sequenced to the learner through a set of learner-initiated or content-initiated navigation events. The system waits until a navigation event is triggered, which is translated into a navigation request and passed to the *Overall Sequencing Process*. The *Overall Sequencing Process* is a set of predefined processes (referred to as behaviors by the IMS SS) that apply the sequencing rules defined in the manifest file to the learner's tacking data and results to one of the following:

- A learning activity is identified for delivery and the LMS launches the corresponding content object.
- No learning activity is identified for delivery. In this case, SCORM does not pose any requirements on the subsequent LMS behavior but recommends providing some sort of guidance to the user and avoid leaving him at a dead-end situation.
- An exception occurs and no learning activity is identified for delivery. In this case, SCORM does not pose any requirements on the subsequent LMS behavior but recommends that the exception is gracefully handled.
- The communication session ends and control is returned to the LMS (i.e. due to an Exit All navigation request).

The *Overall Sequencing Process* and the underlying processes are realized by a set of functions defined as members of the *ContentTreeSCORM* class, as they are only applied to an activity tree.

SCORM Behavior	Function	Description
Overall Sequencing Behavior	overallSequencing()	Controls the execution of the functions described herein.
Navigation Behavior	navigationRequest()	Checks the validly of a navigation request and translates it into termination and sequencing requests.
Termination Behavior	terminationRequest()	Ends the current attempt on the Activity Tree and updates the state of the activity tree.

Rollup Behavior	overallRollup()	Updates the tracking status of a cluster activity is updated according to the current tracking status of its children.
Sequencing Behavior	sequencingRequest()	Traverses the activity tree, applying the sequencing rules defined, in order to identify which activity should be delivered to the learner next. If an activity is identified it issues a delivery request.
Delivery Behavior	deliveryRequest()	Validates a delivery request and if valid, delivers the appropriate content object

Table 4-9: Member functions of the *ContentTreeSCORM* class

Figure 4-6 graphically depicts the internal structure of the overallSequencing() function , as well as its relation with the Activity Tree and the *Tracking Model*.

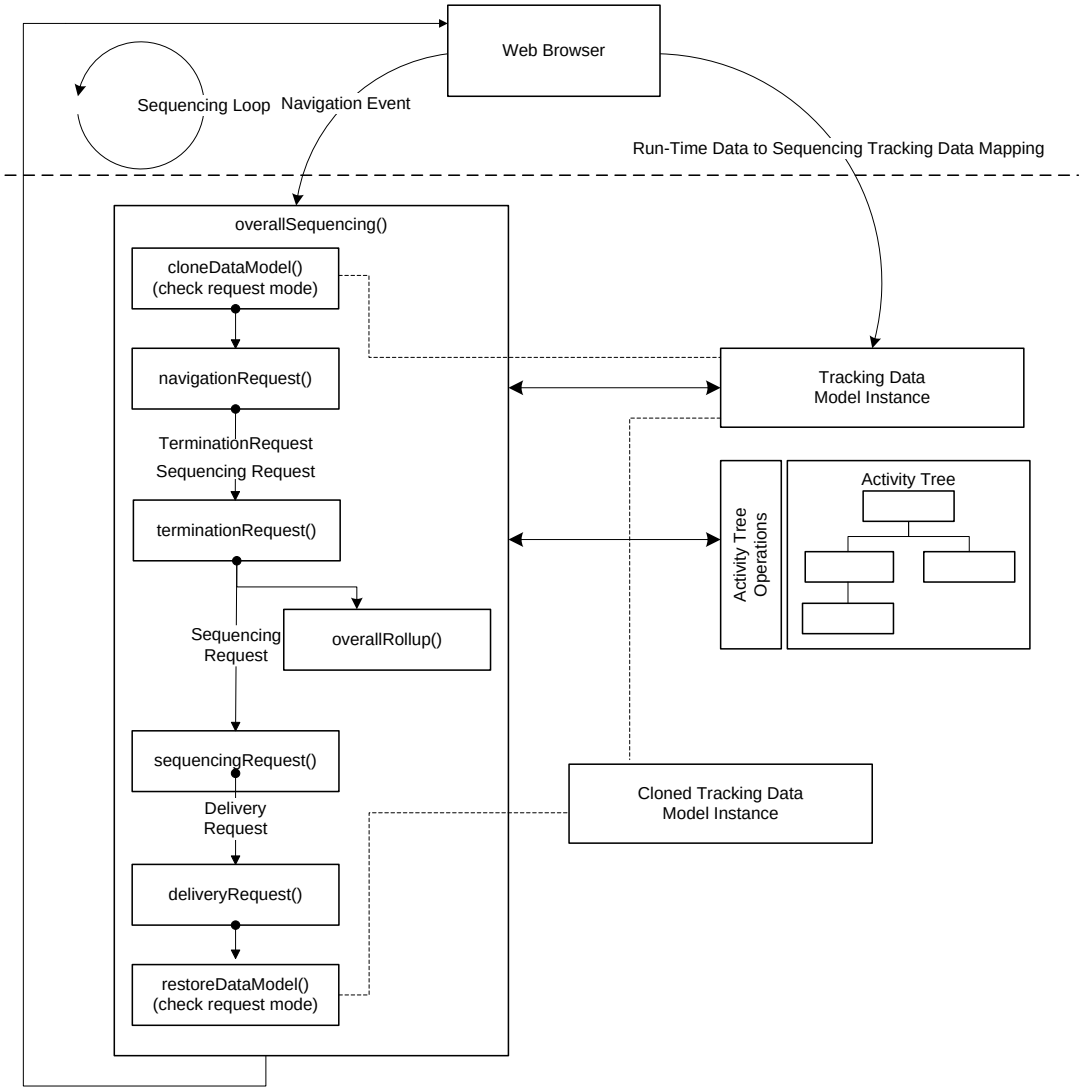


Figure 4-6: Sequencing Loop and internal structure of the overallSequencing() function

The *Overall Sequencing Process* exhibits the behavior of a sequencing loop when a *Sequencing Session* starts until it ends. The *Sequencing Session* is defined as the time from the beginning of an attempt on the root activity of an Activity Tree until its end. The steps of the sequencing loop in the context of our proposed architecture are described in detail below.

1. The learner logs into the LMS and establishes a context with a content organization by selecting one of the available content organizations from the table of contents.
2. The LMS attempts to initiate the sequencing process by automatically issuing a *Start* or a *Resume All* navigation request. The *ContentTreeSCORM* class instantiates. If the previous sequencing session ended due to a Suspend All navigation request, then

the LMS starts the *Sequencing Session* via a *Resume All* navigation request. Otherwise, the LMS starts the *Sequencing Session* via a Start *Navigation Request.* However, the sequencing rules defined may render the *Start Navigation Request* invalid. In this case, it is expected of the learner to manually issue a Choice Navigation request by selecting one of the available activities from the table of contents.

3. The navigationRequest() function, which is the primary entry point of the overallSequencing() function, processes the navigation request. If no errors are encountered the navigation Request is translated to a sequencing Request. The *Sequencing Session* has not yet begun.
4. Based on a sequencing request, the sequencingRequest() function traverses the Activity Tree applying sequencing rules to locate the appropriate activity to deliver to the learner. If no activity is identified for delivery, then the overallSequencing() function stops until another navigation request is issued (Step 9).
5. The deliveryRequest() function determines if the identified activity can be delivered, and if so, the content object associated with the activity is launched. This signals the beginning of the *Sequencing Session*. If the identified activity cannot be delivered, the overallSequencing() function stops. The system waits for another navigation request (Step 9).
6. The learner interacts with the content object.
7. The content object may report values, via the Commit() function of the Run-Time API instance and alter the state of the *Tracking Data Model Instance*. The Commint() function performs an AJAX call to the server-side commit PHP script which updates the various *Tracking Model* elements according to the Run-Time Data to Sequencing Tracking Data Mapping defined by SCORM.
8. A navigation event is triggered by the learner, the content object or the LMS.
9. The LMS translates the navigation event to a navigation request instantiates the *ContentTreeSCORM* class and passes it to the overallSequencing() function.
10. The navigationRequest() function checks the navigation request. If the navigation request has the potential to succeed, it is translated into a termination and a sequencing Request. If the navigation request indicates that the Sequencing Session should end, the learner's content package context is dismissed.
11. If the content package triggered the navigation request by terminating or exiting, it may report additional values that update the *Tracking Model* elements (remember

that the Terminate() function of the Run-Time API makes an implicit call to the Commit() function). The terminationRequest() function ends the attempt on the activity and on its ancestors. The overallRollup() function is called to update the *Tracking Model* for the activity and its ancestors activities within the activity tree.

12. The process repeats beginning at step 4, until the *Sequencing Session* ends.

The overallSequencing() function was developed with two modes of operation – *normal* and *check request*. When called in *normal mode,* the overallSequencing() function, upon exiting, will persist changes made to the *Tracking Data Model Instance*. When called in *check request mode,* the overallSequencing() function creates a copy of the *Tracking Data Model Instance* in order to restore it at its initial state before exiting. No persistence takes place in this mode. This allows evaluating "what if" scenarios for various navigation requests without altering the state of the *Tracking Data Model Instance*. For example, at the second step of the sequencing loop, the LMS attempts to initiate the sequencing process by automatically issuing a *Start* or a *Resume All* navigation request. This is done as follows:

1. The LMS calls the overallSequencing() function in *check request* mode for a *Resume All* navigation request. If no exception occurs, the overallSequencing() function is called in *normal* mode to identify the next activity to be delivered according to where the learner left off during the previous sequencing session.
2. The LMS calls the overallSequencing() function in *check request* mode for a Start navigation request. If no exception occurs, the overallSequencing() function is called in *normal* mode to identify the entry activity of the activity tree.
3. The LMS waits until the learner triggers a *Choice* navigation event.

4.4.4 User Interface Devices

The LMS must provide the ability for a learner to trigger navigation events via user interface devices. SCORM imposes no requirements on the type or style of the user interface presented to a learner at run-time, including any user interfaces devices for navigation. Figure 4-7 illustrates the user interface devices provided to the learner as soon as a context with a content package is established.

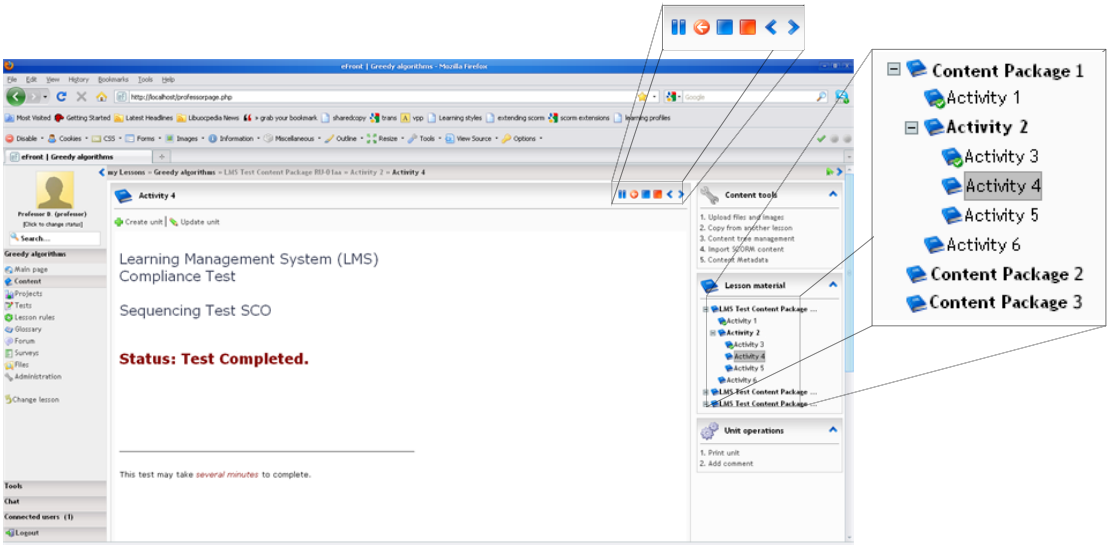


Figure 4-7: SCORM UI Devices

The learner has established a context with *Content Package 1* and is currently experiencing *Activity 4*. The activity titles shown in the table of contents correspond to the name attribute of the item elements in the manifest file. Cluster activities are in bold.

The following table lists the mapping between user interface devices and navigation events.

UI Devices	Navigation Event	Behavior Description
	Continue	This navigation event requests from the LMS to deliver the next logical activity available in the activity tree. This event results in a Continue Navigation Request. In addition an Exit Termination Request is issued if the current activity is still active.
	Previous	This navigation event requests from the LMS to deliver the previous logical activity available in the activity tree. This event results in a Previous Navigation Request. In addition an Exit Termination Request is issued if the current activity is still active.

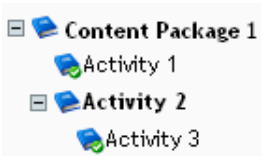
	Abandon	This navigation event requests from the LMS to prematurely or abnormally terminate the current attempt on the current activity. The attempt cannot later be resumed. This event results in an Abandon Navigation Request.
	Abandon All	This navigation event requests from the LMS to prematurely or abnormally terminate the current attempt on the root activity of the activity tree. The attempt cannot be later resumed. This event results in an Abandon All Navigation Request.
	Suspend All	This navigation event requests from the LMS to suspend the current attempt on the root activity of the activity tree. The attempt doesn't end and can be later resumed. Tracking information that already has been recorded will be available when the attempt resumes. This event results in a Suspend All Navigation Request.
	Choose	This navigation event requests from the LMS to deliver a specific activity in the activity tree. This event results in a Choice Navigation Request. In addition an Exit Termination Request is issued if the current activity is still active.
	Exit All	This navigation event requests from the LMS to end the current attempt on the root activity of the activity tree. This event results in an Exit All navigation request.

Table 4-10: UI Devices to Navigation Events Mapping

In Figure 4-7 all of the navigation controls are presented to the learner but this is not always the case. SCORM allows content developers to selectively hide navigation controls from the user interface provided to the learner or disable them using specific declarations in the manifest file. The is*Visible* attribute of the item element specifies whether the corresponding activity in the table of contents should be displayed or not. The *Hide_LMS_UI* element specifies navigation events for which the LMS should not provide the corresponding user interface devices. In addition, SCORM requires from the LMS to provide user interface devices that trigger navigation events whose processing would result in launching a learning resource. If that is not the case, the user interface devices should be visible but disabled. In order to provide such an intelligent user interface, we developed the checkControls() function, as member of the *ContentTreeScorm* class, which checks what navigations controls should be provided to the learner. The checkControls() function operates as follows:

1. Determines which navigations requests are permitted according to the *Hide LMS UI* element.
2. For each permitted navigation request calls the overallSequencing() function in “check request” mode to check if it would result in launching a learning resource.
3. Traverses the Activity Tree checking the *isVisible* attribute of each activity. If the activity is specified as visible, the overallSequencing () function is called in “check request” mode for the corresponding *Choice* navigaton request.
4. An associative array is returned with keys navigation requests and values their states(i.e. enabled, disabled, hidden)

The checkControl() function is called whenever the state of the *Tracking Data Model* changes. As discussed in a previous section, the Commit() function processes the values acquired by the Run-Time data instance model, passes them to a server-side PHP script via an AJAX call, which possibly updates the *Tracking Data Model Instance*. If the *Tracking Data Model Instance* was altered, the PHP script calls the checkControls() function. The array returned is provided via a post-back process to the Commit() function which updates the states of the navigation controls.

4.4.5 Validating the Implementation

Standards serve their purpose when they are implemented correctly, not just when they are implemented. Therefore, ADL releases a test suite along with each SCORM version, containing compliance testing software, procedures and supporting documents for organizations to perform self-testing on learning management systems (LMSs), Sharable Content Objects (SCOs) and content packages. LMS vendors can use the SCORM Test Suite to determine their products' compliance to SCORM, before participating in the ADL Certification Program. The ADL Certification Program is a third-party testing procedure of LMSs organized by authorized ADL Certification Testing Centers and performed using the SCORM Test Suite. If a LMS successfully passes the ADL Certification Program, it is labeled as "ADL Certified", indicating that it successfully implements SCORM's requirements.

The implementation of the proposed architecture was successfully validated using the latest version of the ADL Test Suite (SCORM 2004 4th Edition Version 1.1.1 Test Suite). As far as LMS compliance is concerned, it contains approximately 190 test packages which are used to test the following compliance categories:

- LMS Run-Time Environment (RTE). The LMS shall:
    - Be able to launch SCORM 2004 4th Edition SCO's and assets.
    - Provide and expose an API Instance as a Document Object Model (DOM) object that correctly implements all of the API methods.
    - Implement correct support for all SCORM 2004 4th Edition Run-Time Environment Data Model Elements.
    - Implement correct support for all SCORM 2004 4th Edition Navigation Data Model Elements.
- LMS Content Aggregation Model (CAM). The LMS shall:
    - Be able to "import" and process a SCORM 2004 4th Edition content package.
    - Initialize correctly SCORM 2004 4th Edition Run-Time Environment data model elements based on the information supplied in the manifest file.
- LMS SN (Sequencing and Navigation). The LMS shall:
    - Implement all of the sequencing behaviors defined by the SCORM 2004 4th Edition Sequencing and Navigation (SN).
    - Support all of the SCORM 2004 4th Edition Navigation Data Model Elements.
    - Adhere to navigation user interface requirements defined by the SCORM 2004 4th Edition Sequencing and Navigation (SN).

It is worth mentioning that the SCORM Test Suite proved extremely valuable for understanding some vague parts of the SCORM documentation, as it provides a detailed log for the test actions and the corresponding outcomes. In addition, we tested our implementation against SCORM content created using different SCORM authoring tools (e.g. Reload Editor[18], Articulate Presenter[19] and Articulate QuizMaker[20]). In all cases the content packages behaved as expected. The following subsection describes one of the content packages created with Articulate Presenter and Articulate QuizMaker.

[18] http://www.reload.ac.uk/
[19] http://www.articulate.com/products/presenter.php
[20] http://www.articulate.com/products/quizmaker.php

Example Content Package - Sequential Order

This example exhibits an instructional strategy that forces the learner to visit all activities in their physical order. It involves four activities: two instructional activities, each one followed by an evaluation activity of the knowledge acquired.

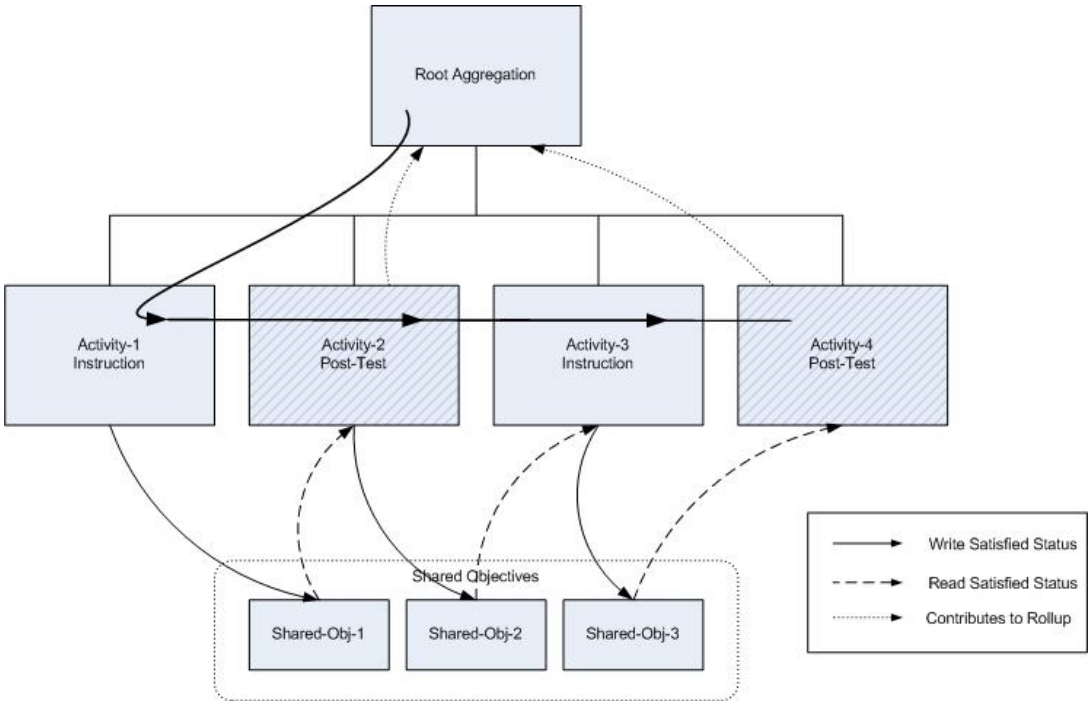


Figure 4-8 Example Content Package (Sequential Order)

When the learner establishes a context with the content organization the LMS automatically issues a *Start* navigation request. The first activity in the Activity Tree is delivered to the learner.

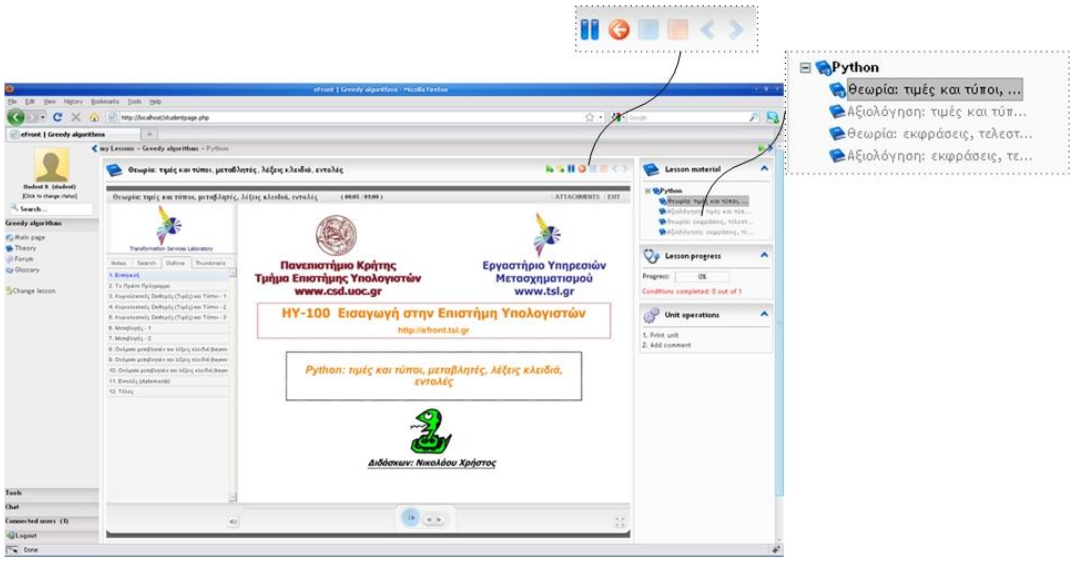


Figure 4-9 Example Content Package (First Activity Delivered)

The *Abandon* and *AbandonAll* navigation controls are disabled due to the following declarations in the manifest file.

```
<adlnav:presentation>
    <adlnav:navigationInterface>
        <adlnav:hideLMSUI>abandon</adlnav:hideLMSUI>
        <adlnav:hideLMSUI>abandonAll</adlnav:hideLMSUI>
    </adlnav:navigationInterface>
</adlnav:presentation>
```

Listing 4-3 Example Content Package (Disabled Navigation Controls)

The *Previous* navigation control has been automatically disabled by the sequencing engine because a *Previous* navigation request would not result to the identification of an activity to be delivered to the learner.

The following *precondition rules* render the *Continue* and *Choice* navigation requests on the second activity invalid. Therefore, the corresponding *Continue* and *Choice* navigation controls are disabled.

```
<imsss:preConditionRule>
    <imsss:ruleConditions conditionCombination="all">
        <imsss:ruleCondition referencedObjective="Activity2-Obj1" operator="not"
condition="satisfied"/>
        <imsss:ruleCondition operator="not" condition="attempted"/>
    </imsss:ruleConditions>
    <imsss:ruleAction action="disabled"/>
</imsss:preConditionRule>

<imsss:preConditionRule>
    <imsss:ruleConditions conditionCombination="all">
        <imsss:ruleCondition referencedObjective="Activity2-Obj1" operator="not"
condition="objectiveStatusKnown"/>
        <imsss:ruleCondition operator="not" condition="attempted"/>
    </imsss:ruleConditions>
    <imsss:ruleAction action="disabled"/>
</imsss:preConditionRule>
```

Listing 4-4 Example Content Package (Precondition Rules - Disabled)

The previous *precondition rules* depend on whether the activity has been previously attempted and on the value of objective *Activity2-Obj1*. This value is aligned with the value of the primary objective of the first activity:

1. The primary objective of the first activity writes its satisfied status to the shared objective *Shared-Obj1*.

```
<imsss:primaryObjective objectiveID = "Activity1-PrimObj">
    <imsss:MapInfo
         targetObjectiveID="Shared-Obj1"
         readSatisfiedStatus="false"
         writeSatisfiedStatus="true"
    />
 </imsss:primaryObjective>
```

Listing 4-5 Example Content Package (Shared Objectives)

2. The objective *Activity2-Obj1* of the second activity reads the satisfied status from the shared objective *Shared-Obj1*.

```
<imsss:objective objectiveID="Activity2-Obj1">
    <imsss:mapInfo
        targetObjectiveID="Shared-Obj1"
        readSatisfiedStatus="true"
    />
</imsss:objective>
```

Listing 4-6 Example Content Package (Shared Objectives)

It is worth noting that if an activity has been attempted once, the precondition rules evaluate to *false*. Thus, the learner has more control over content that has been previously accessed.

The first activity is considered satisfied if ten or more slides of the presentation are viewed. If this is the case, the learner can proceed to the next activity by triggering either a *Continue* or a *Choice* navigation event using the corresponding navigation controls, which are now enabled.

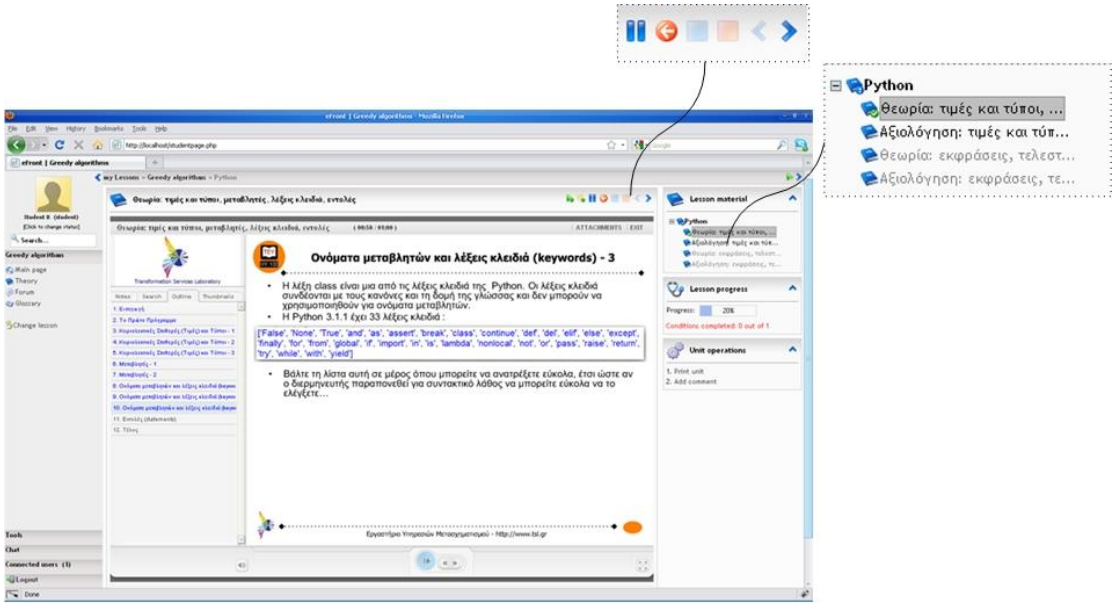


Figure 4-10 Example Content Package (First Activity Satisfied)

The second activity is an evaluation activity of the knowledge acquired during the previous activity. The learner cannot proceed to the next activity before the evaluation activity is satisfied but he is free to go back to the previous activity (by triggering either a *Previous* or a *Choice* navigation event).

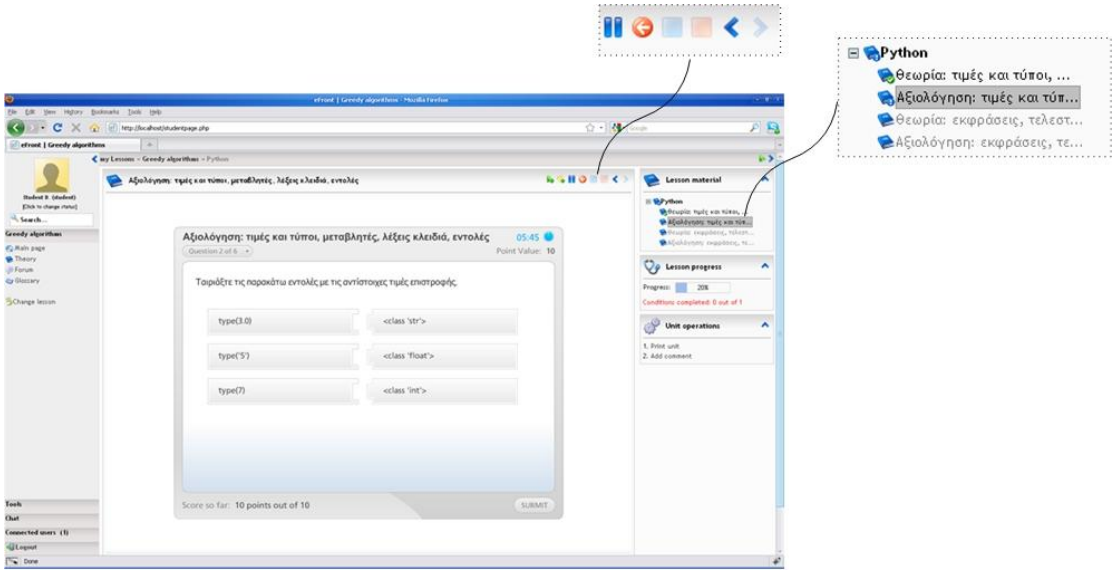


Figure 4-11 Example Content Package (Second Activity Delivered)

Finally, if an evaluation activity is satisfied it will be skipped in the case of *Previous* or *Continue* navigation requests.

```
<imsss:preConditionRule>
    <imsss:ruleConditions conditionCombination="any">
        <imsss:ruleCondition condition="satisfied"/>
    </imsss:ruleConditions>
    <imsss:ruleAction action="skip"/>
</imsss:preConditionRule>
```

Listing 4-7 Example Content Package (Precondition Rule - Skip)

However, evaluation activities can be retaken at anytime through the corresponding navigation requests.

The same sequencing strategy is used for the rest of the activities.

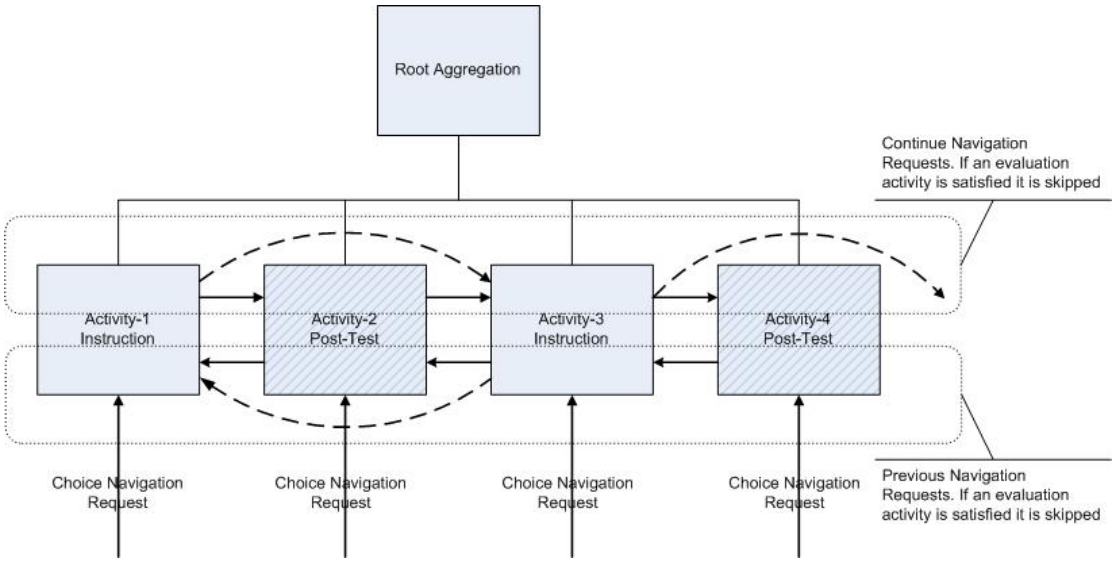


Figure 4-12 Example Content Package (Overall Sequencing Strategy)

The course is considered satisfied if the learner achieves a passing score on the evaluation activities. The rest of the activities do not contribute to rollup. This is specified in the manifest file using the *rollupObjectiveSatisfied* attribute of the *rollupRules* element.

```
<imsss:rollupRules rollupObjectiveSatisfied="false" />
```

Listing 4-8 Example Content Package (Contribute to Rollup)

Because no rollup rules are defined for the cluster activity that have the actions satisfied or not satisfied the sequencing engine will automatically apply the default rollup rules. They would have been written as follows:

```
<imsss:rollupRules>
    <imsss:rollupRule childActivitySet = "all">
    <imsss:rollupConditions>
        <imsss:rollupCondition condition = "objectiveStatusKnown"/>
     </imsss:rollupConditions>
     <imsss:rollupAction action = "notSatisfied"/>
 </imsss:rollupRule>

 <imsss:rollupRule childActivitySet = "all">
    <imsss:rollupConditions>
        <imsss:rollupCondition condition = "satisfied"/>
     </imsss:rollupConditions>
     <imsss:rollupAction action = "satisfied"/>
 </imsss:rollupRule>
<imsss:rollupRules>
```

Listing 4-9 Example Content Package (Rollup Rules)

Chapter 5: **Conclusions and Future Work**

E-learning is becoming increasingly prominent in higher education and corporate training and has already delivered significant results. However, it hasn't reached its true potential as issues such as the interoperability of e-Learning systems and the reusability of learning have not yet been fully resolved. Currently, ADL's work in unifying e-Learning standards under the Sharable Content Object Reference Model (SCORM) is seen as the most promising effort to deal with these issues. SCORM achieved great success with version 1.2 and was quickly adopted from both open-source and commercial LMSs. The following version, however, failed to continue that trend. SCORM 2004 increased in size and complexity, mainly due to the addition of learning content sequencing capabilities as defined by the IMS Global Learning Consortium's Simple Sequencing (SS) specification, and the adoption rate was negatively impacted.

This thesis presented the design and implementation of an architecture for integrating the latest version of the Sharable Content Object Reference Model (SCORM 2004 4th Edition) into a pre-existent learning management system. A great deal of effort was put so that the final implementation reflects accurately the behaviors and functionalities demanded by SCORM. As a result, the implementation successfully passed the excessive testing of the latest version of the ADL Test Suite. Our experience in implementing SCORM leads us to conclude that it is a time consuming process and requires a lot of effort. Consequently, it is reasonable that SCORM 2004 hasn't managed to reach the widespread adoption witnessed in the case of SCORM 1.2. We believe that ADL should reconsider the sequencing part of SCORM and adapt it so that it doesn't pose such a burden to implementers. The importance of learning content sequencing in terms of pedagogy is undeniable, thus removing SCORM's sequencing capabilities is not an option. If that were the case, content developers would return to monolithic SCOs with intra-SCO custom sequencing logic and limited reusability.

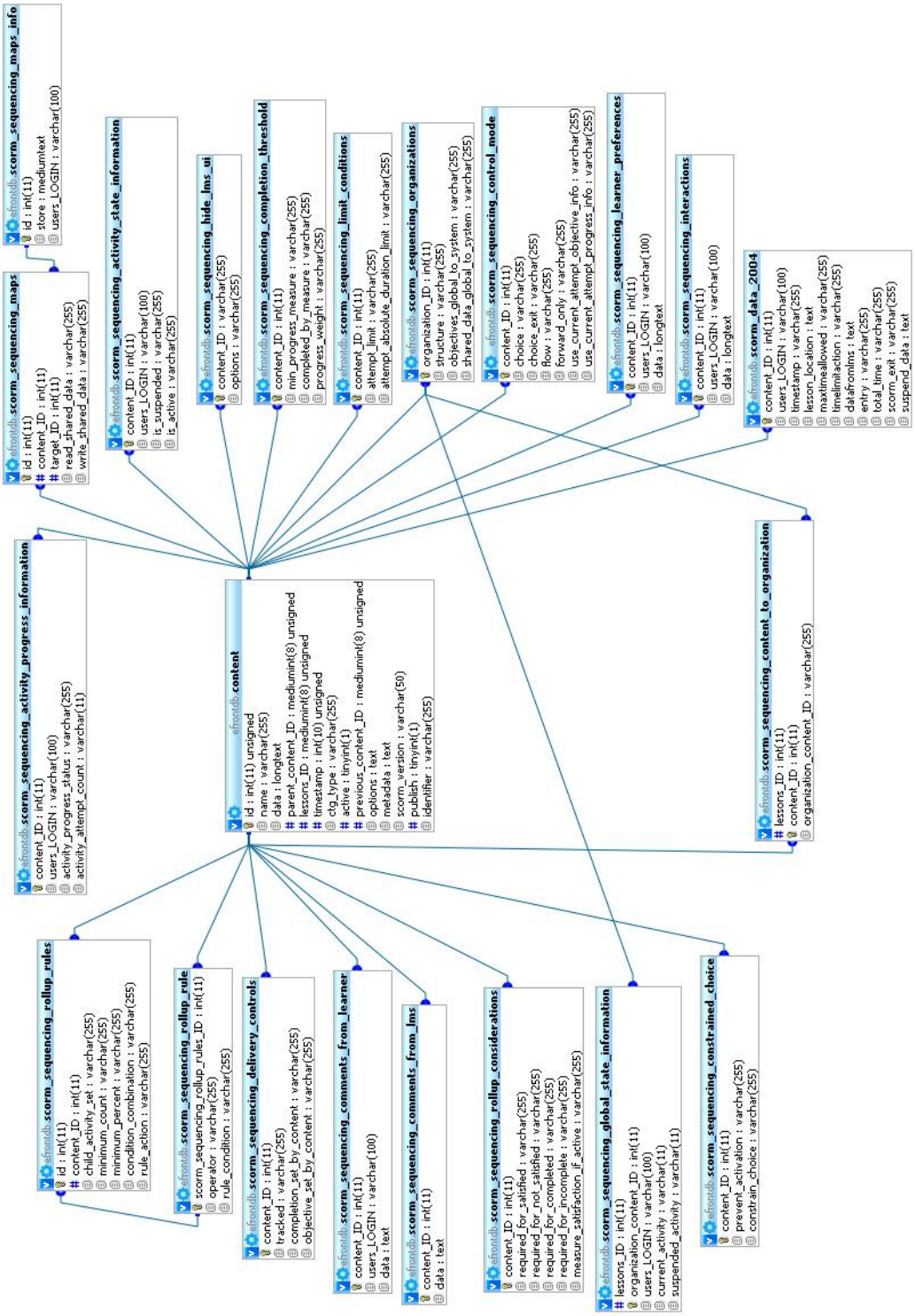
It would be remiss not to point out that there is an increasing interest among LMS vendors for more flexible and cost-effective ways (e.g. service oriented architectures (SOA), web oriented architectures (WOA)) to integrate new services and capabilities into their products. In this sense, many efforts have been made to offer SCORM functionalities as a service, external to the LMS [32] [33] [34]. However, these efforts deal only with parts of the SCORM, not SCORM as a whole. In addition, there are security and performance issues that need to be resolved and the complexity overhead introduced makes it difficult to strictly adhere to the SCORM conformance requirements. Therefore, we believe that ADL should

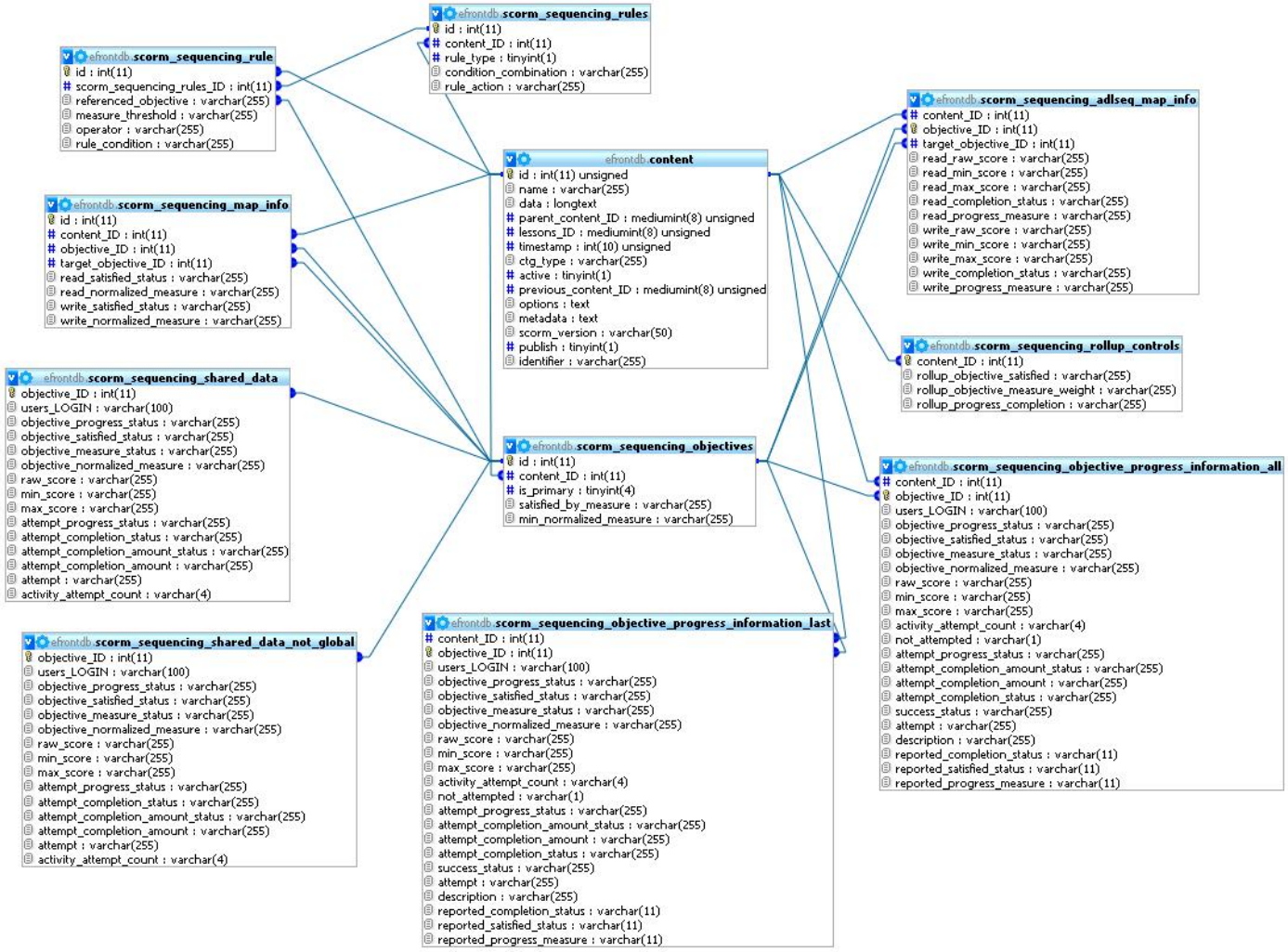
take under consideration these issues and make the proper adjustments to the next version of SCORM, so that it can be applied straightforwardly to service oriented environments.

SCORM has reached a very stable state and is considered nowadays as the de-facto interoperability “standard” for e-Learning systems. However, SCORM is not all inclusive and, since its inception, is continuously evolving to include new features and capabilities. An important issue that hasn’t yet been handled by SCORM is the interoperability of e-Learning systems concerning assessments and their corresponding results. Our work can be the starting point to investigate how a specification like the IMS-QTI can be applied to a SCORM conformant LMS and ultimately be incorporated in SCORM. Other issues that can be investigated in order to propose additions for future SCORM releases are the support for collaborative learning and the formal representation of learner profiles. Currently, SCORM supports only a self-learning model and does not deal with scenarios where learners share a learning experience or work collaboratively to meet a common objective. In terms of modeling learner profiles, SCORM’s CMI data model provides a small set of attributes to capture user preferences and their scope is limited to a single content object.

Appendix

Database Schema




efrontdb.scorm_sequencing_rules
id : int(11)
content_ID : int(11)
rule_type : tinyint(1)
condition_combination : varchar(255)
rule_action : varchar(255)
efrontdb.scorm_sequencing_rule
id : int(11)
scorm_sequencing_rules_ID : int(11)
referenced_objective : varchar(255)
measure_threshold : varchar(255)
operator : varchar(255)
rule_condition : varchar(255)
efrontdb.scorm_sequencing_adlseq_map_info
content_ID : int(11)
objective_ID : int(11)
target_objective_ID : int(11)
read_raw_score : varchar(255)
read_min_score : varchar(255)
read_max_score : varchar(255)
read_completion_status : varchar(255)
read_progress_measure : varchar(255)
write_raw_score : varchar(255)
write_min_score : varchar(255)
write_max_score : varchar(255)
write_completion_status : varchar(255)
write_progress_measure : varchar(255)
efrontdb.content
id : int(11) unsigned
name : varchar(255)
data : longtext
parent_content_ID : mediumint(8) unsigned
lessons_ID : mediumint(8) unsigned
timestamp : int(10) unsigned
ctg_type : varchar(255)
active : tinyint(1)
previous_content_ID : mediumint(8) unsigned
options : text
metadata : text
scorm_version : varchar(50)
publish : tinyint(1)
identifier : varchar(255)
efrontdb.scorm_sequencing_map_info
id : int(11)
content_ID : int(11)
objective_ID : int(11)
target_objective_ID : int(11)
read_satisfied_status : varchar(255)
read_normalized_measure : varchar(255)
write_satisfied_status : varchar(255)
write_normalized_measure : varchar(255)
efrontdb.scorm_sequencing_rollup_controls
content_ID : int(11)
rollup_objective_satisfied : varchar(255)
rollup_objective_measure_weight : varchar(255)
rollup_progress_completion : varchar(255)
efrontdb.scorm_sequencing_shared_data
objective_ID : int(11)
users_LOGIN : varchar(100)
objective_progress_status : varchar(255)
objective_satisfied_status : varchar(255)
objective_measure_status : varchar(255)
objective_normalized_measure : varchar(255)
raw_score : varchar(255)
min_score : varchar(255)
max_score : varchar(255)
attempt_progress_status : varchar(255)
attempt_completion_status : varchar(255)
attempt_completion_amount_status : varchar(255)
attempt_completion_amount : varchar(255)
attempt : varchar(255)
activity_attempt_count : varchar(4)
efrontdb.scorm_sequencing_objectives
id : int(11)
content_ID : int(11)
is_primary : tinyint(4)
satisfied_by_measure : varchar(255)
min_normalized_measure : varchar(255)
efrontdb.scorm_sequencing_objective_progress_information_all
content_ID : int(11)
objective_ID : int(11)
users_LOGIN : varchar(100)
objective_progress_status : varchar(255)
objective_satisfied_status : varchar(255)
objective_measure_status : varchar(255)
objective_normalized_measure : varchar(255)
raw_score : varchar(255)
min_score : varchar(255)
max_score : varchar(255)
activity_attempt_count : varchar(4)
not_attempted : varchar(1)
attempt_progress_status : varchar(255)
attempt_completion_amount_status : varchar(255)
attempt_completion_amount : varchar(255)
attempt_completion_status : varchar(255)
success_status : varchar(255)
attempt : varchar(255)
description : varchar(255)
reported_completion_status : varchar(11)
reported_satisfied_status : varchar(11)
reported_progress_measure : varchar(11)
efrontdb.scorm_sequencing_shared_data_not_global
objective_ID : int(11)
users_LOGIN : varchar(100)
objective_progress_status : varchar(255)
objective_satisfied_status : varchar(255)
objective_measure_status : varchar(255)
objective_normalized_measure : varchar(255)
raw_score : varchar(255)
min_score : varchar(255)
max_score : varchar(255)
attempt_progress_status : varchar(255)
attempt_completion_status : varchar(255)
attempt_completion_amount_status : varchar(255)
attempt_completion_amount : varchar(255)
attempt : varchar(255)
activity_attempt_count : varchar(4)
efrontdb.scorm_sequencing_objective_progress_information_last
content_ID : int(11)
objective_ID : int(11)
users_LOGIN : varchar(100)
objective_progress_status : varchar(255)
objective_satisfied_status : varchar(255)
objective_measure_status : varchar(255)
objective_normalized_measure : varchar(255)
raw_score : varchar(255)
min_score : varchar(255)
max_score : varchar(255)
activity_attempt_count : varchar(4)
not_attempted : varchar(1)
attempt_progress_status : varchar(255)
attempt_completion_amount_status : varchar(255)
attempt_completion_amount : varchar(255)
attempt_completion_status : varchar(255)
success_status : varchar(255)
attempt : varchar(255)
description : varchar(255)
reported_completion_status : varchar(11)
reported_satisfied_status : varchar(11)
reported_progress_measure : varchar(11)

Bibliography

[1] ADL. SCORM Version 1.2 Documentation Suite. [Online]. http://www.adlnet.gov/Technologies/scorm/SCORMSDocuments/Previous%20Versions/SCORM%201.2/Documentation%20Suite%20(SCORM%201.2)/SCORM_1_2_pdf.zip

[2] ADL. SCORM 2004 4th Edition Version 1.1 Documentation Suite. [Online]. http://www.adlnet.gov/Technologies/scorm/SCORMSDocuments/2004%204th%20Edition/Documentation.aspx

[3] IMS Global Learning Consortium. IMS GLC Simple Sequensing Specification. [Online]. http://www.imsglobal.org/simplesequencing/index.html

[4] ADL. SCORM 2004 4th Edition Testing Requirements (TR) Version 1.1. [Online]. http://www.adlnet.gov/Technologies/scorm/SCORMSDocuments/SCORM%202004%204th%20Ed%20V1.1/Documentation%20Suite/SCORM_2004_4ED_v1_1_TR_20090814.pdf

[5] Peter Drucker, "Integrating e-Learning with High Velocity Value Chains," A Delphi Group White Paper, 2000.

[6] Margaret Martinez, "Key Design Considerations for Personalized Learning on the Web," in *Education Technology & Society*, vol. 4, 2001, pp. 26-40.

[7] Margaret Martinez, "An Investigation into Successful Learning: Measuring The Impact of Learning Orientation, A Primary Learner-Difference Variable, on Learning," Instructional Psychology and Technology, Brigham Young University, Dissertation, 1999.

[8] David A Wiley, "Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy," in *The instructional use of learning objects.*, David A Wiley, Ed., 2000, ch. 1.

[9] Wayne Hodgins and Marcia Conner. (2000) Everything you wanted to know about learning objects but were afraid to ask. [Online]. http://www.linezine.com/2.1/features/wheyewtkls.htm

[10] Weller Martin, Pegler Chris, and Mason Robin, "Putting the pieces together: What

working with learning objects means for the educator," in *the Proceedings of the Second eLearnInternational World Summit*, Edimburg, 2003.

[11] Rob Koper, "Combining re-Usable Learning Resources to Pedagogical Purposeful Units of Learning," in *Reusing Online Resources: A Sustainable Approach to eLearning*, A. Littlejohn, Ed. London: Kogan Page, 2003, pp. 47-48.

[12] Daniel R Rehak and Robin D Mason, "Keeping the learning in learning objects," in *Reusing Educational Resources for Networked Learning*, A Littlejohn, Ed. London: Kogan Page, 2003.

[13] National Standards Policy Advisory Committee, "National Policy on Standards for the United States and a Recommended Implementation Plan," Washington, DC, 1978.

[14] Geoff Collier and Robby Robson, "e-LEARNING INTEROPERABILITY," Sun Microsystems, White Paper. [Online]. http://www.sun.com/solutions/documents/white-papers/ed_interop_standards.pdf

[15] Carol Fallon and Sharon Brown, *E-Learning Standards: A Guide to Purchasing, Developing, and Deploying Standards-Conformant E-Learning*. Boca Raton: CRC Press, 2003.

[16] IEEE Learning Technology Standards Committee (LTSC). IEEE Standard for Learning Object Metadata. [Online]. http://ieeexplore.ieee.org/servlet/opac?punumber=8032

[17] Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, Version 1.1. [Online]. http://www.dublincore.org/documents/dces/

[18] IMS Global Learning Consortium. IMS GLC Content Packaging Specification. [Online]. http://www.imsglobal.org/content/packaging/index.html

[19] Aviation Industry CBT Committee (AICC). AICC Guidelines and Reccommendations. [Online]. http://www.aicc.org/docs/tech/cmi001v3-5.pdf

[20] IMS Global Consortium. IMS Learner Information Package Specification. [Online]. http://www.imsglobal.org/profiles/

[21] SIF Association. Schools Interoperability Framework Specifications. [Online]. http://www.sifinfo.org/us/index.asp

[22] IEEE Learning Technology Standards Committee (LTSC). IEEE Standard for Learning Technology - ECMAScript application programming interface for content to runtime services communication. [Online]. http://ieeexplore.ieee.org/servlet/opac?punumber=8972

[23] ISO. (2000) JTC1/SC36 Business Plan. [Online]. http://isotc.iso.org/livelink/livelink/fetch/2000/2122/327993/755080/1054033/2787945/JTC001-N-6296.pdf?nodeid=3912833&vernum=0

[24] Michael Brennan, Susan Funke, and Cushing Anderson, "The Learning Content Management System: A New eLearning Market Segment Emerges.," An IDC White Paper, 2001.

[25] Josh Bersin, "The Four Stages of E-Learning: A Maturity Model for Online Corporate Training," Bersin & Associates, Oakland, 2005.

[26] IMS Global Learning Consortium. IMS GLC Question and Test Interoperability Specification. [Online]. http://www.imsglobal.org/question/

[27] IEEE Learning Technology Standards Committee (LTSC). IEEE Standard for Learning Technology - Data Model for Content to Learning Management System Communication. [Online]. http://ieeexplore.ieee.org/servlet/opac?punumber=9661

[28] Executive Order 13111. (1999, January) Using Technology To Improve Training Opportunites for Federal Government Employees. [Online]. http://www.estrategy.gov/documents/26.pdf

[29] Claude Ostyn, "In the Eye of the SCORM, An introduction to SCORM 2004 for Content Developers," 2007. [Online]. http://www.ostyn.com/standards/docs/Eye_Of_The_SCORM_draft.pdf

[30] Phil Barker, "What is IEEE Learning Object Metadata / IMS Learning Resource Metadata?," CETIS Standards Briefing Series, JISC, 2005.

[31] W3C. XML Base. [Online]. http://www.w3.org/TR/xmlbase/

[32] Costagliola Gennaro, Ferrucci Filomena, and Fuccella Vittorio, "Scorm run-time environment as a service," in *the Proceedings of the 6th international conference on Web engineering*, 2006, pp. 103-110.

[33] "A Web-service oriented framework for building SCORM compatible learning management systems," in *Proceedings of International Conference on Information Technology: Coding and Computing*, vol. 1, 2004, pp. 156-161.

[34] Mariana Karmazi, "A SCORM conformant Sequencing Engine based on WSResource Framework and Principles of Service Oriented Architecture," University of Crete, Heraklion, Master's Thesis, 2009.