Exporting content from content-e to SCORM

Christiaens, A.W.M.

Award date:
2004

Link to publication

Abstract

The assignment, which was the basis for this thesis, is carried out at Turpin Vision. The company Turpin Vision is specialised in the utilization of the possibilities of multimedia in a web based environment. Content-e, the main product of Turpin Vision, is a content management and publishing environment in which you can create, manage, and operate all kind of content.
Turpin Vision aims at the educational and publishers market in the Netherlands with Content-e.

In the global educational market a standard has been set: the Sharable Content Object Reference Model (SCORM). This standard has been developed to create reusable learning objects and to provide a framework for webbased learning. This framework sets requirements on the learning objects and also on the systems using them. These systems are called Learning Management Systems (LMSs) and are oriented at the student. The student learns through this digital system.

More and more LMSs are implementing the SCORM framework. Supporting SCORM as an output format of Content-e, a publication, allows Content-e to deliver content to these LMSs.
The assignment was to define the mapping process from data stored in Content-e to the SCORM format. The assignment has resulted in a definition of the mapping process and a working SCORM publication in Content-e by implementing this process.

The publication has been implemented with the programming language ASP to allow integration with Content-e. It also uses the eXtensible Markup Language (XML) and eXtensible Stylesheet Language Transformations (XSLT).

Preface

This master's thesis report concludes five years of studies within the Information Systems group at the Department of Mathematics and Computer Science at the University of Technology in Eindhoven (TU/e).

The work described in this report has been carried out within the software company Turpin Vision, settled in the Multimedia Pavilion at the TU/e.

I would like to thank Philippe Thiran for his supervision during the project and Egbert Heuvelman for the support at Turpin Vision.

November, 2004
Dries Christiaens

Table of Contents

Index of Figures

Index of Examples

Index of Tables

1. Introduction

Nowadays, in a highly computer oriented world, many people learn by studying materials found on the Internet or Intranet of their organization. Materials can be made available in an electronic learning environment, allowing people to interactively view these materials. This environment is called a Learning Management System (LMS).
Many courses exist to educate and train people. Each LMS has its own views on how to support and utilize these courses. This has the following drawback: every course has to be adapted to the underlying LMS, which involves time and money spent each time the course needs to be changed. Specifications for content have to be met to make content in courses more reusable. Content needs to be structured and described as stated in these specifications.
Several organizations are involved in the process of developing propositions and specifications for learning materials. If learning management systems comply with these specifications, courseware can be imported into these systems without any adjustments.
A standard has been devised on how content materials should be packaged and described, in order to be used in different systems. This standard is called Shareable Content Object Reference Model (SCORM) and it is described in [Dodds, 2001]. This standard also requires specific functionality to be implemented in a LMS as part of the framework of SCORM. This functionality is necessary to handle SCORM conforming packages in the LMS.

Turpin Vision has built a product called Content-e. Content-e is an Learning Content Management System (LCMS), a system to create courseware and make it available to other systems, such as a LMS. This thesis will discuss how content stored in Content-e can be exported to SCORM packages. The exported packages can then be imported into a LMS.

This chapter will discuss the systems involved in the research, the goal to be reached and the current situation. The problem description is given, which is addressed by this thesis. The chapter ends with a description of the structure of the thesis.

1.1. Involved systems

SCORM packages are imported into a LMS. The definitions LMS and LCMS are explained in the following section.
Content-e is the system from which the content has to be exported. SCORM is the standard to which content needs to be exported.

1.1.1. LCMS and LMS

A large difference exists between LCMS and LMS, although their names are quite similar.
In a LMS teachers are able to track the progress of a student and the learning environment can adapt the structure of the courseware to the needs of a student. Students learn courses offered by the LMS. The primary focus is on the student. Examples of LMSs are Blackboard and N@tschool.

In a LCMS learning objects can be created, manipulated and delivered to other systems, the primary focus is on the learning object. The objective of a LMS is to deliver content to the right person at the right time.
In [Valkenburg, 2004] the concepts LCMS and LMS are discussed in more depth.

The application Content-e, which contains the materials to be exported, will now be discussed.

1.1.2. Content-e

Content-e is a system to create and manage content, such as text and multimedia, in a web based environment. It is implemented as a client-server model. The client is a program installed on a computer and sends requests to a server. This is a different program, sends back a response. The server can even be installed on a different computer,
Content-e supports multiple users working on the same document and has got a very thorough permissions system. It is possible to export the content to various formats, such as Word and PDF. An export is called a publication in Content-e. In the remainder of this thesis, whenever 'publication' is mentioned, it is used in the context of Content-e, and represents an export.
The system is completely web based and doesn't require an installation of software on the computer of the user.
In the following sections the entities of which Content-e consists are discussed.

In Figure 1 the structure of Content-e is explained. Content-e uses of a browser as client to communicate with the server side of Content-e. This side contains the webserver, which communicates with a database server. In order to properly display the Content-e application, the browser will download files from the server to enhance its functionality.
After a person has entered content in the author system, he can view it in this same system. When he wants to use this content in a different system, a publication can be made. All materials belonging to the chosen room, desk or book will be collected and put in a separate place on the server. The publication process will return an URL, a link to these materials. Anybody who has access to the management system of Content-e or to the publications list available from the portal page of Content-e, can then download these materials to his own system. They can also view them from the server, if they can be loaded in a webbrowser.

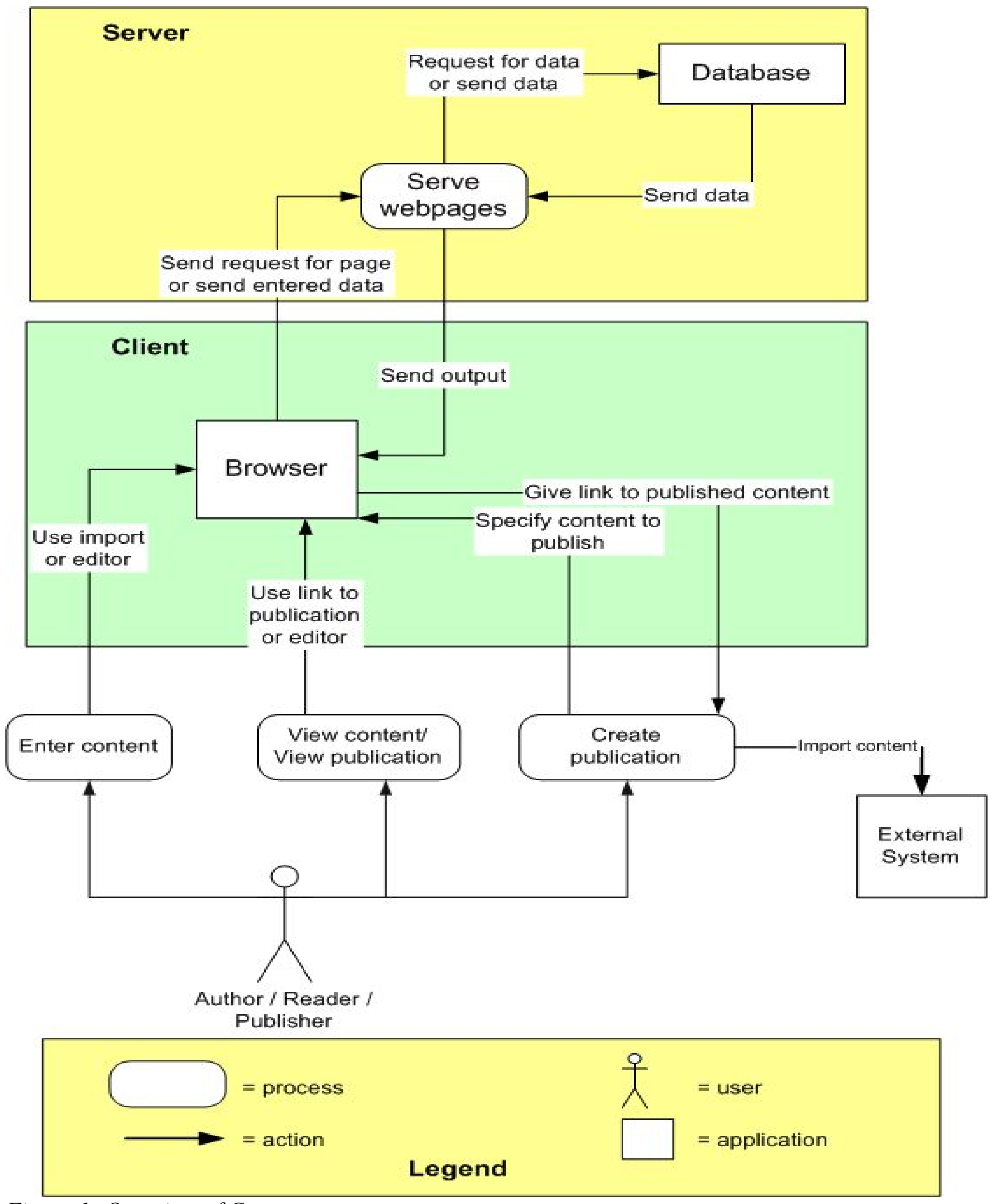

Server
Request for data or send data
Database
Serve webpages
Send data
Send request for page or send entered data
Client
Send output
Browser
Give link to published content
Specify content to publish
Use import or editor
Use link to publication or editor
Enter content
View content/ View publication
Create publication
Import content
External System
Author / Reader / Publisher
= process
= user
= action
= application
Legend

Figure 1: Overview of Content-e

The various parts of Content-e will now be explored, starting with some brief descriptions of the technical components. Concepts and metaphores are discussed after these descriptions. The section ends with some screen shots of the application to give an impression of the application.

1.1.2.1.Browser
Content-e requires a person to use a computer with a browser installed on it and a working Internet connection. A browser enables the view of pages published on the World Wide Web.

1.1.2.2.Webserver
A client needs to connect to a server, in a client-server model. The webserver is part of the server side in Content-e. A webserver delivers webpages to a client, when requested. It can handle input of a user entered on a webpage in a browser.

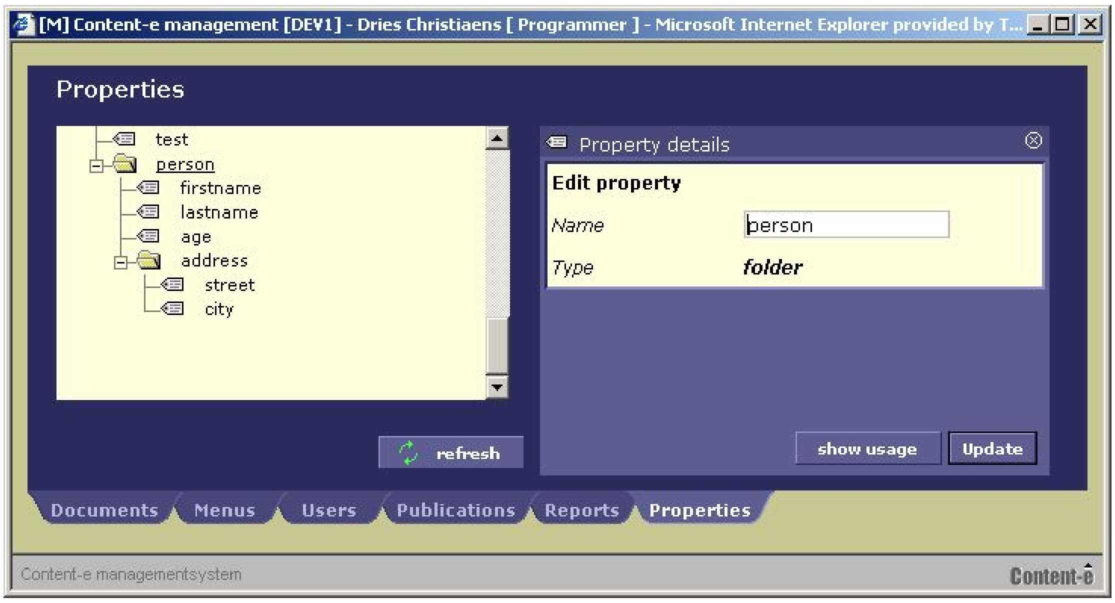
1.1.2.3.Database
A webserver can deliver data to a client, but in order to store data and retrieve data the use of a database is of great importance. A database server handles requests from programs, such as a webserver or other applications, and depending on the request, updates or retrieves data.
Retrieval of data can be done with the use of a query language like SQL. SQL stands for Structured Query Language and it is a standard language for accessing and manipulating databases, such as Microsoft SQL Server and Oracle.
A relational database manipulates only tables and the result of all operations are also tables. The tables are sets, which are themselves sets of rows and columns. You can view the database itself as a set of tables.

The concepts and metaphores used in Content-e are explained next to give an impression about what Content-e actually does.

1.1.2.4.Concepts and metaphores
Content-e has several metaphores such as menus, documents and paragraphs. Menus have menu items, each menu item can refer to a document or to a different menu. Documents consist of one or more paragraphs, each can be commented on, or edited. A paragraph contains content like text, multimedia etc.
To make it more understandable to the user working with the application, there are three different menu types:
- Room: collection of desks, can also contain references to other rooms.
- Desk: collection of books.
- Book: collection of documents.

Content-e also has the metaphore excursion. With an excursion it is possible to view presentations, video clips, images and websites in Content-e. Excursions can be attached to paragraphs.

Another metaphore in Content-e are properties. Properties can be added to menus, documents and paragraphs. Each property has a type and can be assigned one or more values.
These properties are represented as a tree structure, where the leaves have values assigned to them.



Edit property

Name: person

Type: folder

Figure 2: Properties in Content-e

In Figure 2 the tab sheet, available in the Management System of Content-e, is shown in which it is possible to create, edit and remove property structures in Content-e. Each property has a type and can be nested into another property. In this figure a property 'person' has been created. It consists of child properties 'firstname', 'lastname', 'age' and 'address'. The last mentioned property contains the child properties 'street' and 'city'.
Values are not assigned to the properties here, this can be done in the 'Documents' and 'Menus' tab sheets in the Management System or in the Author System, which are discussed in the following sections.
Properties can be used in a publication module or by Content-e itself, in the filter. The filter is a set of properties with assigned values, which allows documents and menus to be distinguished. Documents and menus, which don't apply to this filter, are either greyed out or made invisible with an active filter in the Management System.

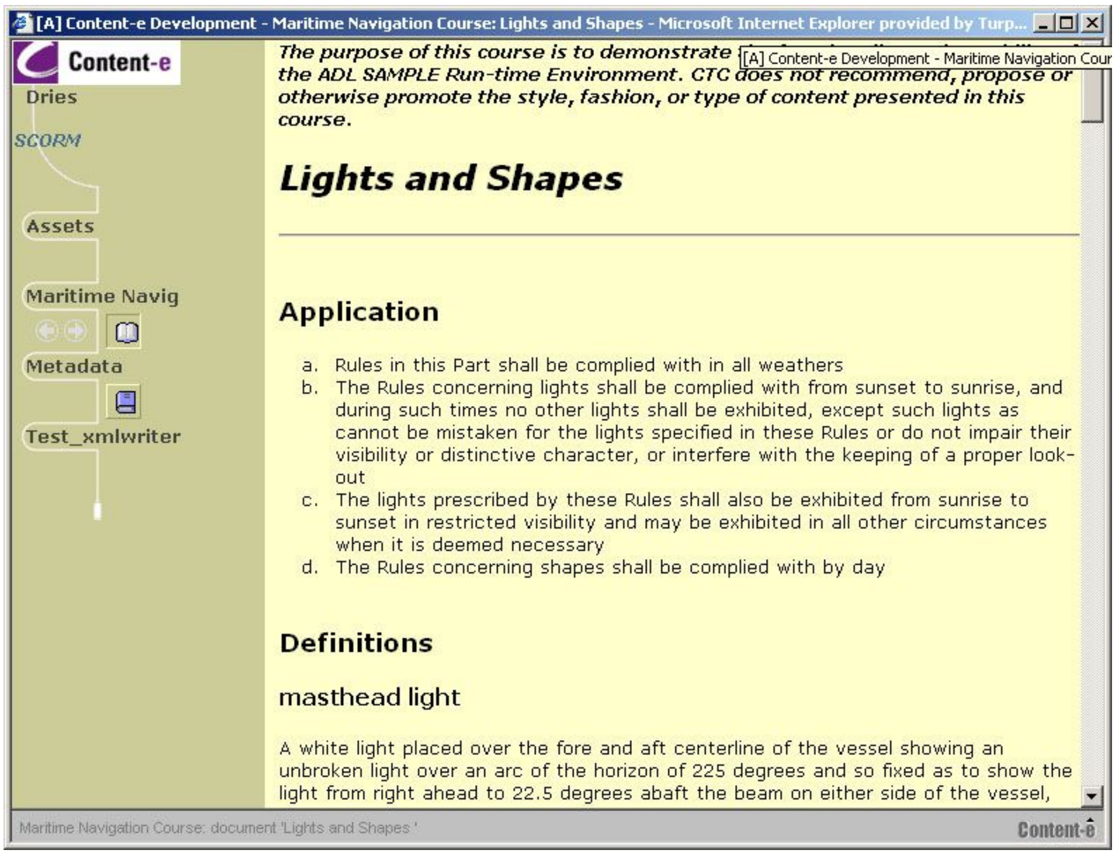
Another facility is the hyperlink system. It is possible to create normal hyperlinks to websites in a paragraph in a document, but also links to paragraphs of other documents in Content-e can be created.

Publications of rooms, desks or books can be generated to make their data available to other systems than Content-e. Publications provide an export of data.

Menus, documents and paragraphs can be manipulated in both the Author System and the Management System. These sytems are available through links in the portal of a Content-e system. A portal provides user access to the various parts of Content-e.

1.1.2.5.Content-e Author System

In this system a default room or desk is displayed, which is set for the person accessing Content-e and is called the start menu.
In a desk the books opened by the user are visualized by icons. Clicking on such an icon lets the user jump to the last recently opened document in that book. In Figure 3 the author system of Content-e is shown with a open book 'Maritime Navigation Course' and a document 'Lights and Shapes' in that book. It is possible to edit the content of this document if the user has enough permissions to do so. With less permissions the user is only able to view the document or it will be invisible to him.
The permissions depend on the role the current user has in Content-e and whether or not the creator of the document has allowed the current user to view or manipulate the document. Analogously menus have been provided with access rights.


[A] Content-e Development - Maritime Navigation Course: Lights and Shapes - Microsoft Internet Explorer provided by Turp...
Content-e
Dries
SCORM
Assets
Maritime Navig
Metadata
Test_xmlwriter
Maritime Navigation Course: document 'Lights and Shapes '
Content-e

The purpose of this course is to demonstrate [A] Content-e Development - Maritime Navigation Cour *the ADL SAMPLE Run-time Environment. CTC does not recommend, propose or otherwise promote the style, fashion, or type of content presented in this course.*

Lights and Shapes

Application

a. Rules in this Part shall be complied with in all weathers
b. The Rules concerning lights shall be complied with from sunset to sunrise, and during such times no other lights shall be exhibited, except such lights as cannot be mistaken for the lights specified in these Rules or do not impair their visibility or distinctive character, or interfere with the keeping of a proper look-out
c. The lights prescribed by these Rules shall also be exhibited from sunrise to sunset in restricted visibility and may be exhibited in all other circumstances when it is deemed necessary
d. The Rules concerning shapes shall be complied with by day

Definitions

masthead light

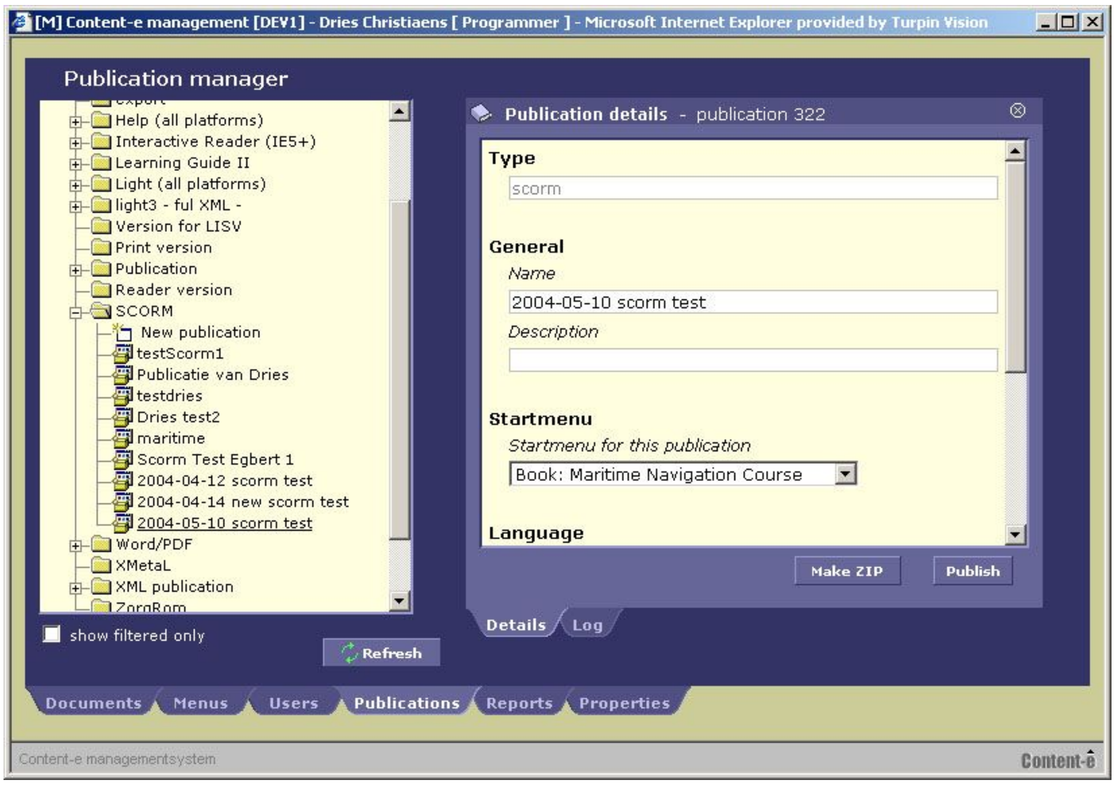
A white light placed over the fore and aft centerline of the vessel showing an unbroken light over an arc of the horizon of 225 degrees and so fixed as to show the light from right ahead to 22.5 degrees abaft the beam on either side of the vessel,

Figure 3: Content-e author system.

1.1.2.6.Content-e Management system

In the Management system it is possible to export the data supplied in the author system to other systems, with the creation of publications. In Figure 4 the details of a publication, the SCORM publication to be devised, are shown.
Other tasks which can be performed in this part of Content-e are the creation and/or manipulation of documents and menus.

User management has also been incorporated into this system. If the current user has enough permissions, he can view and change user details. Some user details are quite important such as the role the user has in Content-e. Different roles offer different general capabilities. Currently a standard role and a administrator role have been defined. The last role allows the creation of new users in Content-e and update of details of all users.



Publication details - publication 322

Type
scorm

General
Name: 2004-05-10 scorm test
Description:

Startmenu
Startmenu for this publication: Book: Maritime Navigation Course

Language

Figure 4: Content-e management system

Several standard publications such as the Word and PDF publication exist in Content-e. It is also possible to create a very custom publication format, only to be used by one customer, if needed.

Turpin Vision is interested in producing a publication, which can be used in several external systems. SCORM is a format to be used in different learning systems, it was designed for this reason.

In the following section the SCORM standard will be explored.

1.1.3. SCORM

The Department of Defense of the United States ordered an initiative to make courses usable in different web-based learning management systems. The need for a specification was very high, due to the time and money involved in the adaptation of courses to every learning system used by the military.

This initiative is called the Advanced Distributed Learning (ADL[1]) organization. ADL devised the Sharable Content Object Reference Model (SCORM) standard, which uses specifications of the ARIADNE, AICC, IEEE LTSC and IMS organizations. These organizations are all concerned with creating or using specifications describing and structuring content or the interaction of systems with content. Data conforming to specifications can be constructed in a program and used in several others. The definition 'content' in the context of SCORM is used to describe digital learning materials.

Several aspects exist to which content, usable in different systems, have to comply:

- Reusability: content should be context independent, so it can easily be used again, for instance in different courses.
- Accessibility: content can be identified and located when needed.
- Durability: content doesn’t have to be changed, if the environment's system is modified.
- Interoperability: the same content will function in different systems.

These aspects together form requirements of the framework of SCORM.
A LMS supporting SCORM implements a runtime system, allowing usage of this content to be tracked. Also the interaction between content and the LMS, in which it is used, is specified.
Using this framework in a LMS requires the implementation of a single interface in the LMS instead of various specifications defined by different vendors producing learning materials.
Applications producing learning materials also have to support only one standard instead of different specifications defined by various LMSs.

SCORM has been and still is developed with feedback from different areas of the e-learning market. Recommendations from users and producers have been incorporated to make the standard useful to both sides of the market.

Several concepts and their level in SCORM will now be discussed:

- Manifest
- Asset
- SCO
- Aggregation
- Metadata
- LMS interaction
- Runtime system in a LMS

1.1.3.1.Manifest

A manifest describes what content materials are used in the package and what their relations to each other are. Dependencies on files are explicitly mentioned in the manifest. It defines the structure in which documents are organized, in essence it is the heart of the SCORM package. Without the manifest the package is useless.

1 See http://www.adlnet.org

1.1.3.2.Asset
An asset is the lowest conceptual level in SCORM. Assets in SCORM are 'simple' resources such as images, video clips etc. or more 'complex' like a group of webpages. They are completely passive, meaning that they don't have interaction with a learning management system (LMS).

1.1.3.3.SCO
One level higher than assets, SCOs can be found.
A SCO (Shareable Content Object) can consist of zero or more assets and is the smallest unit traceable by a LMS. Each SCO should act independently and referencing a different SCO has to be avoided to maintain reusability of the SCO, one of the aspects of SCORM. The dependencies of a SCO, such as the used assets, are explicitly mentioned in the manifest. Courses consisting of SCOs are used in a web-based learning environment, therefore the SCOs have to be written in a format like HTML, supported by a browser. HTML is a language to write web pages with.

1.1.3.4.CAM
A Content Aggregation Model (CAM) is at the highest conceptual level in SCORM. An aggregation is a set of learning materials belonging to the same unit, such as a course or module. An aggregations consists of multiple SCOs and defines a structure of these SCOs. A manifest describes the CAM.

1.1.3.5.Metadata
Each conceptual level in SCORM can have metadata attached to it. Metadata is used to describe objects, and is of great importance whenever content is transported to a different system or when the content is stored in a repository. In a large repository it is easy to loose objects. Metadata provides a search on the object with requested properties in the repository.

An analogy can be made with a library. The library is a representation of the repository, objects are represented by books. Without a description of the book, the solution is going through all books and match the name of the requested book, which can be a cumbersome task, if there are thousands of books. This will only work if you know the name of the book. By creating a library card for every book, with a description of the book, like category, name of the writer and other useful information, the search for a book can be done based on sections of this description. This will also return multiple books of interest, if the descriptions of these books match the criteria of the user. The library cards are representations of metadata. The Dublin Core[2] group has made a technical specification for metadata used by many. IMS has made extensions to this specification to provide a metadata set for the educational environment. Instead of using RDF, a language which is used to describe resources on the Internet to implement metadata, IMS defined their own implementation.
An example of a RDF file is given in Example 1.

2 See http://www.dublincore.org

```
<?xml version="1.0" ?>
<!DOCTYPE rdf:RDF PUBLIC "-//DUBLIN CORE//DCMES DTD
2002/07/31//EN"
    "http://dublincore.org/documents/2002/07/31/dcmes-xml/dcmes-
xml-dtd.dtd">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description
rdf:about="http://www.ilrt.bristol.ac.uk/people/cmdjb/">
    <dc:title>Dave Beckett's Home Page</dc:title>
    <dc:creator>Dave Beckett</dc:creator>
    <dc:publisher>ILRT, University of Bristol</dc:publisher>
    <dc:date>2002-07-31</dc:date>
  </rdf:Description>
</rdf:RDF>
```

Example 1: A RDF file

1.1.3.6.Runtime system in the LMS

A package of SCOs is imported in a LMS, after which a user can view the contents of the package through the LMS. The LMS decides which SCO gets loaded, based on the manifest describing the SCOs located in the package. A SCO can't dictate which SCO should be loaded next, this can only be done by the LMS. A requirement of the runtime system is that only one SCO at as time can be loaded by LMS, a SCO has to be unloaded before a new SCO can be loaded.
The LMS creates a navigation structure, for instance a menu, according to the manifest, and thereby allows SCOs to be viewed.
Variables in the SCORM runtime system of a LMS can be manipulated by SCOs as mentioned in the previous section, only after the SCO has been loaded by the LMS.

1.1.3.7.Interaction with the LMS

The method of communicating with the LMS has been predefined in SCORM, this has to be done with ECMAScript, implemented in browsers with Javascript/JScript. SCORM defines methods for retrieval and setting of variables by SCOs, making it possible to collect information about the progress of the student working with the course. These variables are made available by the LMS through an API, an Application Program Interface.

1.1.3.8.Description of a SCORM package

In Figure 5 a representation is given of the contents of a SCORM package.
This package contains a manifest describing the structure of the package, content materials in the form of HTML documents, metadata files in XML format and assets in binary or text format.

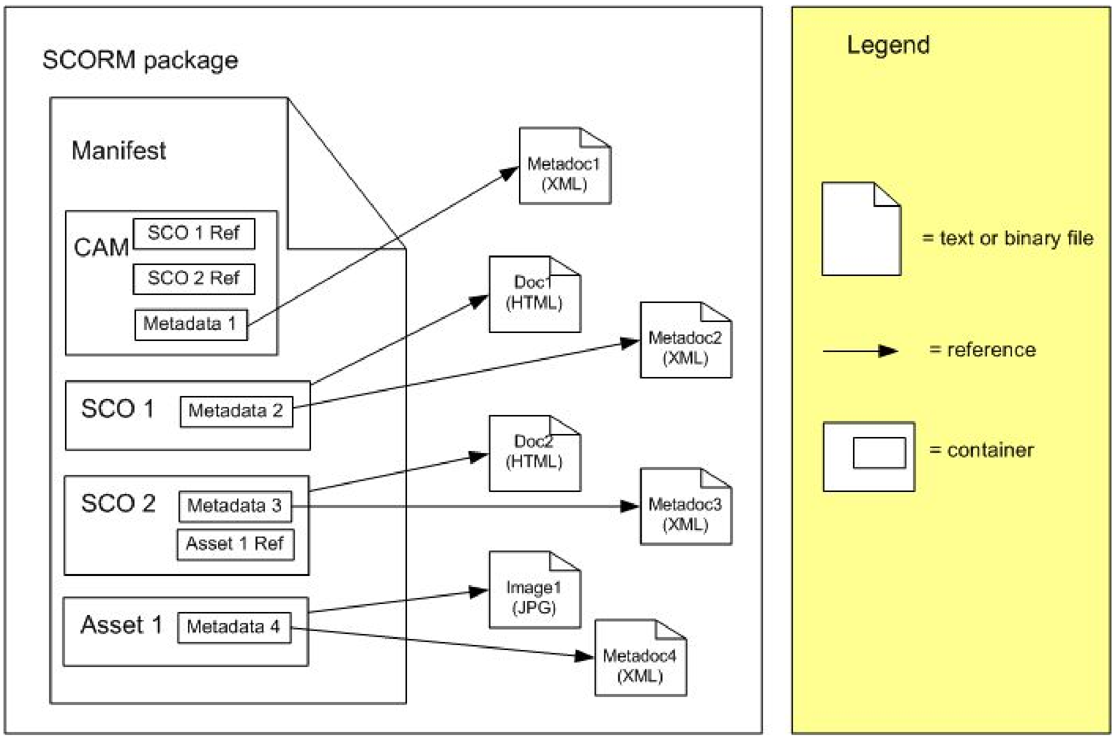


Figure 5: Overview of a SCORM package

In Figure 5 an overview is shown of a SCORM package.

The manifest of a SCORM package is used in a SCORM supporting LMS to create a menu structure. This structure allows a user to select a item, corresponding to a SCO. A 'hierarchial' structure in the manifest is displayed as a tree-like structure in the LMS. The 'choice' structure is represented by 'forward' and 'previous' options in the LMS.

In Figure 6 a example is given of the interaction of a SCO with a LMS. Communication is always started from the SCO, the LMS can send a response back if needed.

If prerequisites of a SCO have been defined, they have to be fulfilled by the user in order to view the SCO. In the specifications of the mainstream version of SCORM this facility has not been explored very thoroughly, and this has been addressed by the lastest version of SCORM, SCORM 2004.

The mainstream version of SCORM is 1.2, used by several products. Its successor has been released in January of this year, but as with all new specifications it will take a while before the new version is incorporated in commercial products. In the rest of this thesis version 1.2 of SCORM has been considered.

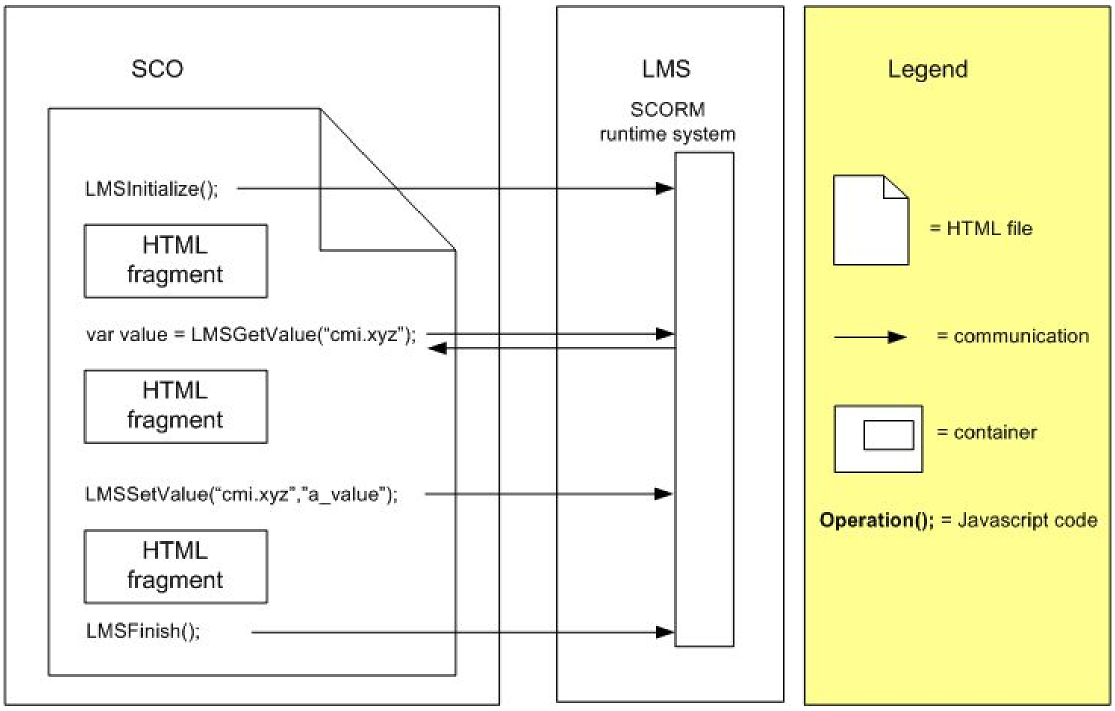


Figure 6: Interaction of SCO with LMS

After exploring the systems involved in the research, the goal will now be discussed.

1.2. Goal of the thesis

The goal of this research is to create a SCORM publication, export data from Content-e to a format conforming to SCORM.
This will make content from Content-e available to other systems supporting SCORM.

The process of creating content, publishing it as SCORM, and importing it into a LMS is visualized in Figure 7. An author creates or edits content in Content-e, and afterwards makes a publication, a Content-e export, to the SCORM format. This content, in packaged form, can be imported in a LMS, to allow students to learn from the content, through the LMS.
This export of data can also be represented by a mapping process from data of Content-e to SCORM.

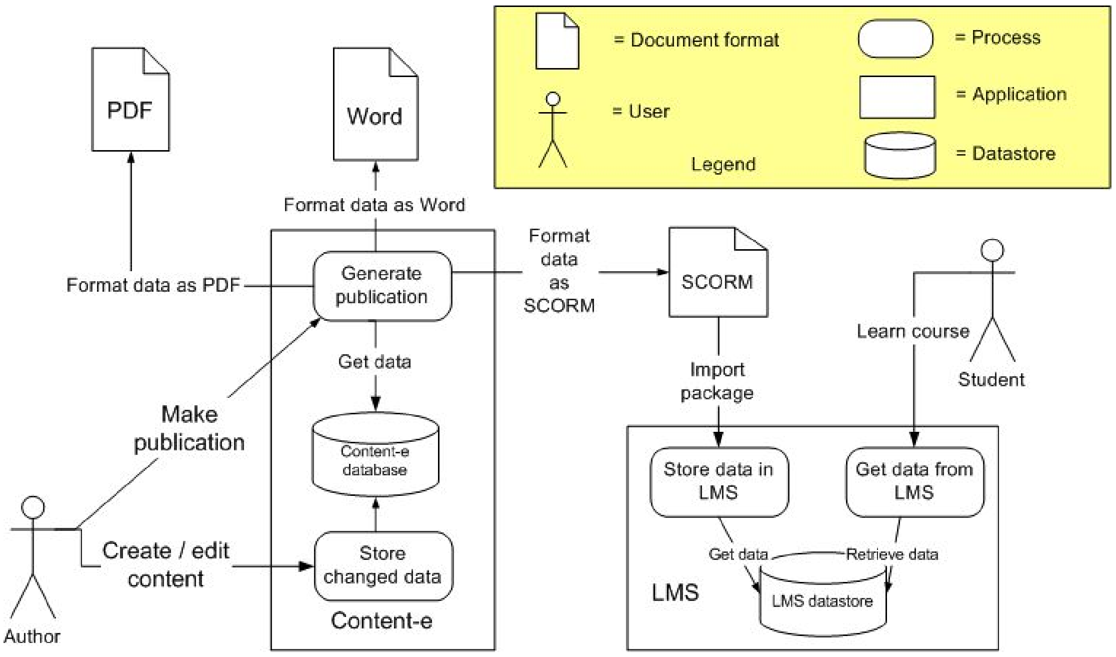


Figure 7: Overview of export of data from Content-e to a LMS

1.3. Description of the current situation

Content-e is able to export to various formats. A first attempt to a SCORM publication has been made, but it needs to be explored further. The already existing publication module for SCORM produced a publication of a book and its documents. The manifest describing the aggregration of the SCOs was only provided with some basic elements.
It also lacked the notion of metadata and it also didn't incorporate the minimal mandatory communication signals, which a SCO should give to a LMS. A redesign of this publication needs to be made. Also the SCORM standard needs to be explored further.

The problem description will now be given, which is addressed in the remainder of this thesis.

1.4. Description of the problem

The main problem to be researched in this thesis: How to export content in the underlying relational database of Content-e to a package of files conforming to the SCORM standard?
This should be done as much as possible with the facilities offered by Content-e.

SCORM defines requirements which should be met by learning materials, and it also requires some functionality of LMSs. The remainder of this thesis considers these requirements, because the goal is to export data to a SCORM conforming package of learning materials. The discussion of how LMSs should implement the required functionality for SCORM is not a topic in this thesis.

The problem can be divided into smaller problems:

- At what level should the data be retrieved?
- How is the data retrieved from the Content-e database?
- How can this data be transformed to a SCORM conforming package?
- How should interaction with the LMS, in which the package will be placed, be implemented in the documents of the package?

These questions will be explored further in the following sections.

1.4.1. At what level should the data be retrieved?

Several options exist in the level of data retrieval. Content-e has the concept of rooms, desks, books, documents and paragraphs. What would be a logical starting point to export the data from?

1.4.2. How is the data retrieved from the database?

Data retrieval can be done with the use of SQL, the query language for databases or with the use of already existing programmed objects. Do the objects offer enough information, making explicit SQL queries and creation of custom-made objects unnecessary?

1.4.3. How to transform this data to a SCORM conforming package?

The manifest in a SCORM conforming package has to conform to a specification, implemented by a XML Schema. XML Schemas are discussed in the next chapter. Metadata describing the content in the package also have to conform to a specification, again in the form of a XML Schema.
How should the generated files comply with these specifications?

1.4.4. How to implement interaction with the LMS?

Interaction with the LMS needs to be done with Javascript, currently the only language permitted to handle communication, according to the SCORM standard. The SCOs have to be provided with Javascript calls to set and get the necessary values from the LMS. SCOs have the responsibility of searching for the API adapter, the connection with the LMS.
How can the SCOs be provided with these Javascript calls?

1.5. Document structure

In chapter 2 the technologies will be discussed which are involved in the different systems. First the different technologies in Content-e are mentioned. They are followed by a discussion of technologies used in the transformation process, which starts with the content exported from Content-e, and ends up with the content in a format conforming to SCORM. Finally technologies used in SCORM will be reviewed.

Chapter 3 will show the methodology used for solving the main problem of how to export data from the Content-e database to a package conforming to SCORM. Methodology describes the utilized methods in a systematic way.

The end situation is SCORM conforming package. Concepts from SCORM will have to matched with Content-e to meet this end situation. Once the matching has been defined, a mapping process from Content-e to SCORM can be devised. Difficulties encountered in the mapping process and their solutions are discussed. Answers are given on the questions of the main problem stated in chapter 1.

In chapter 4 the mapping process defined in chapter 3 is implemented. An intermediate structure has been constructed to apply transformations for files conforming to SCORM. A different approach for assigning values to properties in Content-e has been implemented. Difficulties in the implementation in each step of the mapping process are discussed and possible solutions are provided and implemented.

In chapter 5 conclusions are made about the process of exporting data from Content-e to a SCORM conforming package. A working SCORM publication in Content-e has been made. Remarks about future improvements in the SCORM publication in Content-e are mentioned.

2. Technology

2.1. Introduction

There are several technologies involved in the process of accomplishing an SCORM publication from Content-e.
Technologies used in Content-e, in the transformation process, and in SCORM are discussed.

2.2. Technologies used in Content-e.

In this section the different technologies which are used in Content-e will be discussed.

2.2.1. Database

First of all, the data is stored in a relational database produced by Microsoft, the Microsoft SQL Server 2000, which supports stored procedures. Stored procedures act as little programs, manipulating or selecting data on the database server, thereby avoiding having to output data from the relational tables to server side programs, which would have to perform all kinds of tests, updates and other actions. Stored procedures run on the database server and are fast for this reason. They can most often return the desired set of data required by the server side program, ready for immediate use without another step of changing the data with the use of SQL, a querying language for databases mentioned earlier. Stored procedures are used in ASP objects of Content-e and also in publications.

2.2.2. Programming language server-side

To present data or inspect it, a server side program is needed. ASP is used in Content-e as technology to accomplish this. ASP stands for Active Server Pages and was invented by Microsoft. It has to be processed by a webserver, it can't run standalone.
ASP pages enable programmers to build dynamic pages to present data in a webpage or to deal with user input in a browser. The ASP files in Content-e are written in the scripting language Javascript.

2.2.3. Webserver

As already mentioned in the previous section ASP needs a webserver to be useful. A webserver is able to serve pages to a client, most often a browser. Content-e uses the Internet Information Server (IIS) from Microsoft for this task.

2.2.4. Browser

To connect to the webserver and show a user-interface to the user a browser is needed. A browser can display HTML and XML. With the use of plugins the number of supported formats can be increased.
To run Content-e the Internet Explorer from Microsoft is required. Technology embedded in this browser is used in Content-e.

2.2.5. Programming language client-side

To enhance the capabilities of a browser the scripting language Javascript is used. It allows the processing of data on the computer of the user before sending it to the server and also provides other useful functionality in a browser.

2.2.6. Markup language client-side

HyperText Markup Language (HTML) is used to display web pages, of which the user interface of Content-e consists. This language is frequently used to create web pages on the Internet, because it has been quite easy for people to work with. Also a large amount of tools are available to create and maintain webpages.

2.3. Technologies used in the transformation process

Starting point of the transformation process is the content in the Content-e database. The data is transformed into XML. Different techniques involving XML will be discussed now, such as XML itself, but also XSLT and schemas for XML.

2.3.1. XML

The eXtensible Markup Language (XML) is a structured markup language, which allows data to be transported from one system to another. Since it is only data either system can do whatever they want with it, such as creating their own documents out of it. XML data can be made from a programming language with the use of bindings. Microsoft for instance provides bindings for XML, implemented in the language JScript, to create and manipulate data in XML format. XML is quite easy to read and edit, because it is written in plain text and in a structured format. However to display the data in a more convenient way to people, it can be transformed to a different format with the use of XSLT.

```
<?xml version=”1.0” ?>
<lom id=”1”>
  <general>
    <title>
      <langstring>
        Conduct of Vessels in any Condition of Visibility
      </langstring>
    </title>
    <language>en-US</language>
    <description>
      <langstring>
        Discusses general rules of operation for vessels on
        inland waters. Topics discussed include: Look-out, Safe
        Speed, Collision, Channels, Traffic Separation.
      </langstring>
    </description>
    <keyword>
      <langstring>Vessel</langstring>
    </keyword>
  </general>
</lom>
```

Example 2: A XML file

Basically a XML file can be represented as a tree. Every element is a node in the tree, represented by a opening tag, ***<element name>***, and a closing tag, ***</element name>***. It has zero or more child nodes. A XML file has only one root, in Example 2 this is the element 'lom'. Every element can have a value: **<language>**en-US**</language>** means that the element 'language' has the value 'en-US'. The element 'lom' also has an attribute 'id' with value '1'. An attribute can be used to provide information that is not part of the data. The text in Example 2 is structured and quite easy to read, these are properties of XML. For more information on XML the website of W3C can be visited.

2.3.2. XML Schema

XML data is represented by a structure, which can comply with a predefined file, called a schema. A schema defines which structures are possible for the XML file. Examples of schemas are DTD (Document Type Definition) and XSD (XML Schema Definition).
Several validation tools which accept a XML file and its schema have been developed and can be either downloaded for free or bought on the Internet and can be of great assistance in creating and verifying valid XML files.
Defining the possible structures with a schema, makes it possible to transport data from one system to another. Each party involved knows what structures to expect. This will make the processing task of XML data easier.

In Example 3 a simplified XML Schema for metadata is shown. It defines a definition for a XML representation of a 'general' section. This section contains a title and optional descriptions.
The title is mandatory: in the line **<xsd:element name="title" type="xsd:string"/>** the attributes 'minOccurs' and 'maxOccurs' have been omitted. They specify how often the element at least and at most appears. If these attributes are omitted, they have the default value 1. The title is a text represented by the type **xsd:string**.
Zero or more descriptions are possible, as defined by the attributes 'minOccurs' and 'maxOccurs' in the line **<xsd:element name="description" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>**. A description is also represented by the type **xsd:string**.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="generalType">
  <xsd:sequence>
     <xsd:element name="title" type="xsd:string"/>
     <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="general" type="generalType"/>
```

Example 3: A simple XML schema

2.3.3. XSLT

The eXtensible Stylesheet Language Transformations (XSLT) is a transforming language which requires a XML file as input and is able to output to different formats, depending on what is specified in the XSLT file.
The language is based on matches. Whenever a section of a XML file is matched with a section in the XSLT, a transformation is applied.
Often the output format is HTML to view data in a browser, or it is XML, to make it suitable for a different system. XSLT is written in XML, thereby removing the necessity of learning another language for users, if they already understand XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <lom id="1">
    <xsl:apply-templates select="./property"/>
  </lom>
</xsl:template>

<xsl:template match="property">
  <xsl:element name="{./name}">
    <xsl:value-of select="./value"/>
    <xsl:apply-templates select="./property"/>
  </xsl:element>
</xsl:template>

</xsl:stylesheet>
```

Example 4: A simple XSLT file

The code in Example 4 shows a simple XSLT file and contains two matchings.
If the root of XML file is matched, a new element 'lom' with attribute 'id' is created. The attribute has value '1'. If the root has a child element 'property', matching continues.
Assume an element 'property' exists with a child element 'name', which has the value 'language'. This element 'property' contains another element 'value' containing the value 'en-US'.
The second matching in Example 4 creates a new element 'language' with the value 'en-US' with this assumption. If the element 'property' contains a child element 'property', the matching continues.

In Example 5 a XML file has been provided to illustrate the use of the XSLT file in Example 4. Assume this XSLT has been saved as a file with the name 'example_property.xsl'. If the XSLT file is applied on the XML file, an output is generated as provided in Example 2.

In this situation a XML file is used as input and the produced output is also a XML file.

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="example_property.xsl"?>
<property>
  <name>general</name>
  <property>
    <name>title</name>
    <property>
      <name>langstring</name>
      <value>
        Conduct of Vessels in any Condition of Visibility
      </value>
    </property>
  </property>
  <property>
    <name>language</name>
    <value>en-US</value>
  </property>
  <property>
    <name>description</name>
    <property>
      <name>langstring</name>
      <value>
         Discusses general rules of operation for vessels on
         inland waters. Topics discussed include: Look-out, Safe
         Speed, Collision, Channels, Traffic Separation.
      </value>
    </property>
  </property>
  <property>
    <name>keyword</name>
    <property>
      <name>langstring</name>
      <value>Vessel</value>
    </property>
  </property>
</property>
```

Example 5: A XML file containing a XSLT reference

2.4. Technologies in SCORM

A SCORM package can be put in ZIP format, a popular compression method. It bundles all files into one large file, making transportation easier, for instance over the Internet. The manifest file, describing the package, is in XML format and has to comply with XSD files, made available by the ADL organization. Both XML and XSD have been discussed in the previous section.

Learning materials viewed by a person, are available in HTML, the most popular format for writing web pages on the Internet. Information about these learning materials, metadata, are available in XML format, and also have to comply with a XSD file, again made available by ADL.

2.4.1. Javascript

A different section of the framework of SCORM is the interaction with the LMS, implemented with Javascript. This language is used in the learning materials. However it needs to connect to the LMS, some kind of connection point in the LMS should be available. This point is known as the API adapter.

2.4.2. API adapter & Runtime system of the LMS

An API adapter allows Javascript calls to the LMS, to retrieve and set data. This adapter can be written as an Java applet, making the adapter available to SCOs in a browser. However the implementation of the API adapter hasn't been dictated by SCORM, a API adapter made in Flash can also be possible. The API adapter is part of the LMS.

First the LMS will load a SCO. In this SCO two Javascript calls should be made. The first call is to inform the LMS, that the SCO is loaded and ready for interaction. This is done with the **`LMSInitialize()`** command. The SCO can now retrieve data with **`LMSGetValue()`** and set data with **`LMSSetValue()`**. After the SCO is finished communicating with the LMS, it will send the **`LMSFinish()`** command. Any subsequent call to the LMS after this command results in an error. This is visualized in Figure 8.
Each call to the LMS results in a return value from the LMS, to be interpreted as an acknowlegdement of the call sent to the LMS.

A user is able to select a different SCO based on the available SCOs provided by the LMS.
The interesting property of the runtime system of the LMS is that data set by one SCO can be used in another SCO. It is also possible to let the variables in the LMS be inspected by a tutor for example, to see how a student has responded to questions in a SCO. The implementation of these variables is not specified to be of a certain language. Each vendor of a LMS, wishing to support SCORM, should only have to respect the specifications of what variables should be available to SCOs.
The SCORM Runtime System will assure that at most one SCO can be loaded any time. It will provide a user interface for selecting SCOs, based on the manifest to which the SCOs belong. The communication with SCOs is done through the API adapter, discussed in the previous section.

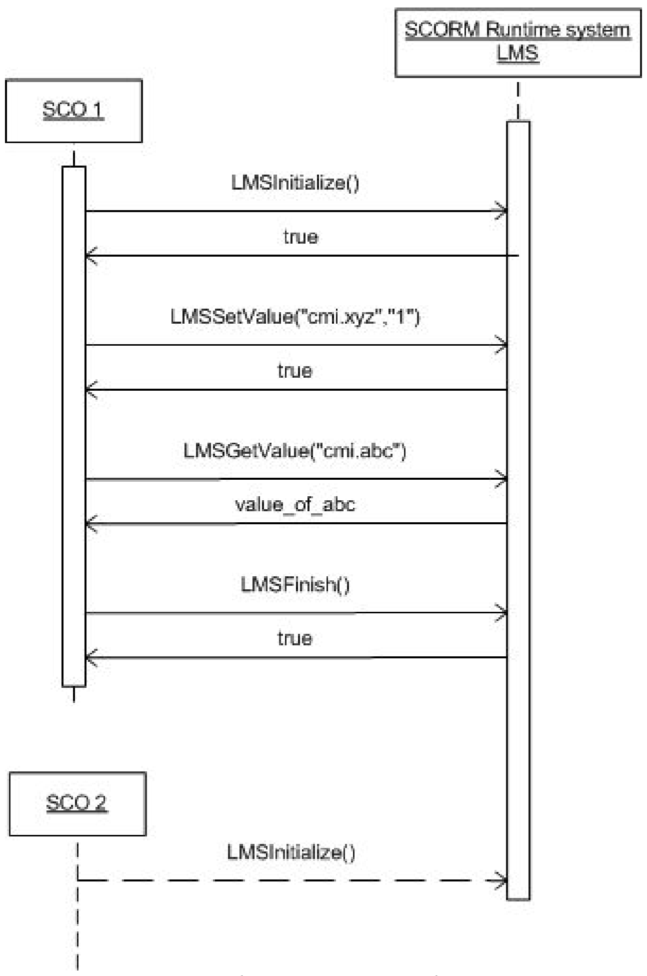


Figure 8: Communication between SCOs and a LMS

3. Methods for matching Content-e to SCORM

3.1. Introduction

The main problem of this thesis is how to export data from Content-e to SCORM. This problem is visualized in Figure 9.

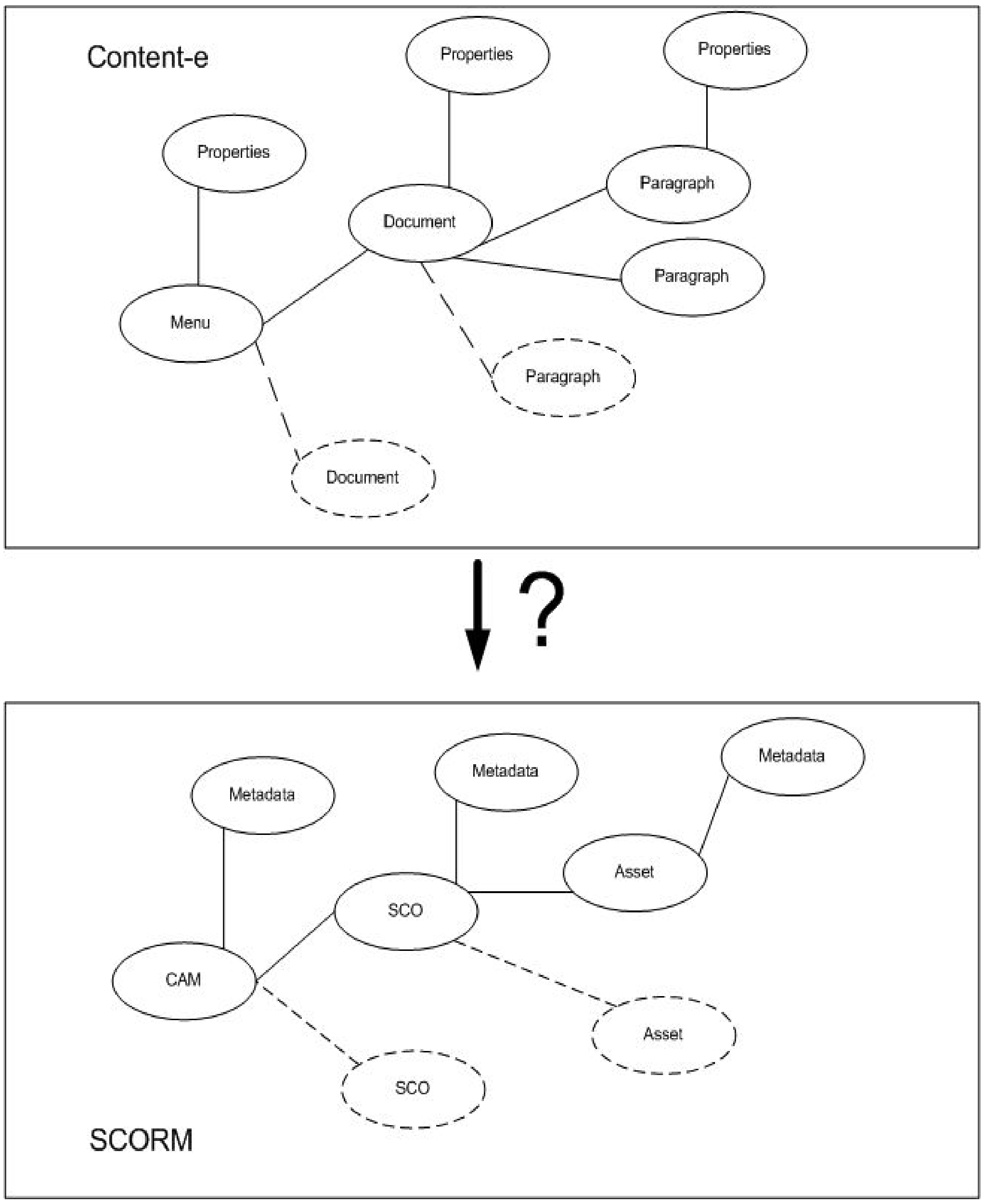


Figure 9: How to map Content-e to SCORM?

This problem can be addressed by taking an approach, which includes two tasks:
1. The first task to be performed is to find which elements in Content-e match concepts in SCORM.
2. The second task is to create a mapping process for the matches found in the first task. If representations for SCORM in Content-e don't exist they have to be added during the process if possible, to ensure that data is formatted in conformance with the SCORM standard.

The four questions of the main problem stated in chapter 1 are answered by performing the first and second task.
The first task should answer the question:
- At what level should the data be retrieved?
- How should interaction with the LMS, in which the package will be placed, be implemented in the documents of the package?

The second task should give answers to the two remaining questions:
- How is the data retrieved from the Content-e database?
- How can this data be transformed to a SCORM conforming package?

Next the first task defined above is dealt with.

3.2. Establishing matches between Content-e and SCORM

Elements in Content-e need to be found, which are good representations for concepts in SCORM. Once the matches have been found, the first problem stated in chapter 1, "At what level in Content-e should the data be retrieved?", can be solved.

Content-e	***SCORM***
Menu/document	Content Aggregation Model (CAM)
Menu/document/paragraph	Shareable Content Object (SCO)
Reference to a file, such as image or video clip, in a document	Asset
Properties	Metadata
-	Calls to the LMS

Table 1: Concepts in SCORM and their possible representations in Content-e

In Table 1 an overview of concepts in SCORM and their representations in Content-e are shown. For CAM and SCO there exist multiple options in Content-e. Assets and metadata have an obvious matching in Content-e. The last concept 'Calls to the LMS' doesn't have a counterpart in Content-e. A short discussion and the best choice for each of the concepts is given in the following sections.

3.2.1. CAM

As can be seen in Table 1 a menu or document are options to act as source for matching an aggregation. In SCORM an aggregation is a set of learning materials belonging to the same unit, as mentioned in chapter 1.
The counterpart for an aggregation in Content-e is an entity which allows learning materials to be grouped together. A menu and a document fit this description. A room houses one or more desks, a desk contains one or more books in Content-e. Documents belong to a book. A room, desk and room are menu types as mentioned in chapter 1. A document is used to group paragraphs together.
Menus and documents exist as possible representations for a CAM in Content-e.
The choice for the CAM representation will be postponed until a representation for a SCO has been defined. This will eliminate a few of the CAM representations, after which a best choice can be made for the CAM representation.
The following section discusses what options exist for a SCO representation in Content-e and which of these options is best suited for the main problem.

3.2.2. SCO

A SCO can consist of one or more web pages. As already mentioned in the introduction a SCO is the lowest level at which a LMS can track a SCO. It should be an independent piece of information, in order to be reusable. The different levels of content in Content-e, menus, documents and paragraphs, are discussed as possible options to represent a SCO in the following sections.

3.2.2.1.Paragraph as representation

Using a paragraph as a SCO representation is actually quite illogical. A SCO is expected to be an independent piece of information. A paragraph in Content-e doesn't fit this functionality however. It is always part of a document and it is perfectly normal to reference to other paragraphs in the same document. Showing a representation of a paragraph on its own in a LMS doesn't make sense, it depends on other paragraphs to be understood by a user and should be shown with accompanying paragraphs of the same document. This fails the requirement of being reusable, acting independently.

3.2.2.2.Menu as representation

The lowest level of a menu is a book in Content-e consisting of multiple documents. If a book was chosen to represent a SCO, this would correspond with a SCO consisting of multiple webpages. The LMS will not be able to track the progress of a separate webpage in this SCO. A SCO is the lowest level traceable in a LMS. However it should quite often be possible to track the progress of which webpages have been read by a student working with the LMS.
This is a serious disadvantage for selecting a book as a SCO. Higher levels than a book will give this same disadvantage.

3.2.2.3.Document as representation

If a document is considered as a SCO, the problem would turn the other way around. Every webpage is tracked by the LMS and it is not possible to create a SCO of multiple pages. If a document is chosen as a SCO representation, it should be able to act independently of other documents. Referencing other documents by creating links to them within Content-e in the Author System prevents this requirement to be fulfilled and should therefore be avoided.

It is however perfectly valid to include content from a different document, a feature provided by Content-e, as this doesn't breach the last mentioned requirement. Including content from a different document copies a structure of paragraphs from this document to the document requiring the inclusion.

3.2.2.4.Best choice for the SCO representation

The options for the SCO representation have been discussed and their implications have been explored.
The choice is actually between a menu and a document, a paragraph can't be regarded as a serious candidate, due to the reusability requirement, which it doesn't meet.

So a decision has to be made between matching a menu or document from Content-e to a SCO. The ideal SCO representation would be able to choose a document or menu in the appropriate situation, but for now a fixed representation is used.
Looking at the available examples of SCORM packages on the Internet has led to the conclusion that a SCO is most often implemented as a single webpage. A page in Content-e is represented by a document, so a logical conclusion would be to match a document in Content-e to a SCO.

3.2.2.5.Implication for the CAM representation

Matching a document to a SCO implies that the representation of a CAM in Content-e should be at a higher level than documents. Documents needs to be aggregated into a unit.
Thereby the option mentioned earlier of using documents as a CAM representation can be dismissed and only menus are left as valid options. A type of menu needs to be designated as container of the documents. The first choice would be books. A book contains documents and can define a structure the documents like an aggregation defines the structure of SCOs.

3.2.3. Asset

In Content-e a document can contain resources in its paragraphs e.g. images, excursions to presentations or video clips. This corresponds exactly with the description of assets in SCORM. The mapping of resources to assets implies files to be copied from Content-e to the SCORM package. However resources in Content-e are not administrated like menus or documents, they don't exist as objects.
A resource is considered to be an asset if it is referenced more than once implying it is being reused.

3.2.4. Metadata

In Content-e properties exist, which can be attached to menus, documents and paragraphs. These properties can be used in a publication from Content-e. They seem ideal to match to metadata used in SCORM.
In SCORM it is possible to have metadata defined for assets. However due the remark stated in the previous section, Content-e doesn't provide a simple way of attaching properties to resources. Metadata are not required according to the SCORM standard. However if metadata is used, it should conform to specifications of the SCORM standard. Properties will be used in Content-e wherever possible to represent metadata.

The properties can be constructed in a tree structure which closely resembles the metadata structure in SCORM. By closely resembling the metadata structure the amount of work involved in transforming properties to metadata will be small. However the preparation of the property tree structure involves quite some work, as well as supplying values for these properties.
Creating this tree structure and supplying it with data actually introduces a preparation step in the process of exporting data from Content-e to SCORM.

3.2.5. Calls to the LMS

In a SCORM supporting LMS, changing values of variables and reading the values of variables in SCORM runtime system of the LMS can be done with SCOs. To do this, certain points in the learning materials should contain the calls to the LMS. These calls should be represented in Content-e with some kind of markers, which can be inserted in the text in documents. However currently the use of markers hasn't been implemented in Content-e and are quite specifically of use for the SCORM publication.
These markers should have the requirement that they don't interfere with publications of Content-e to other formats than SCORM, since the markers won't have any use in a Word publication, for instance. A first simple step at implementing these markers can be to present the author of the learning materials with a list of calls, out of which one can be selected.
Implementing markers is a topic for future research, as most documents in Content-e currently don't require user interaction, which was one of the purposes of the SCORM Runtime System in a LMS.

Calling the start and end signal of the LMS by a SCO can done without markers. Setting the status of a SCO to 'complete' can also be done while unloading the SCO to let teachers track which pages have been viewed by a student.

This answers the question “How should interaction with the LMS, in which the package will be placed, be implemented in the documents of the package?”. LMS calls can be implemented with markers, but to make the calls of any use, interaction in the documents is required. Due to the absence of interaction in most documents in Content-e at the time of this research, markers aren't implemented in Content-e. The design and implementation of markers is a topic of future research.
The basic LMS start and end calls don't require markers to be inserted in documents and they are generated in the mapping process.

3.2.6. Matches between Content-e and SCORM

All representations in Content-e presented in Table 1 have now been set, based on decisions made in the previous sections and they are shown in Table 2.

Content-e	***SCORM***
Menu (type book)	Content Aggregation Model (CAM)
Document	Shareable Content Object (SCO)
Reference to a file, such as image or video clip, in paragraphs of a document	Asset
Properties	Metadata
-	Calls to the LMS

Table 2: Concepts in SCORM and their chosen representations in Content-e

The first task stated in the beginning has been carried out and the question at which level the data should be retrieved from Content-e can be answered with the book level.

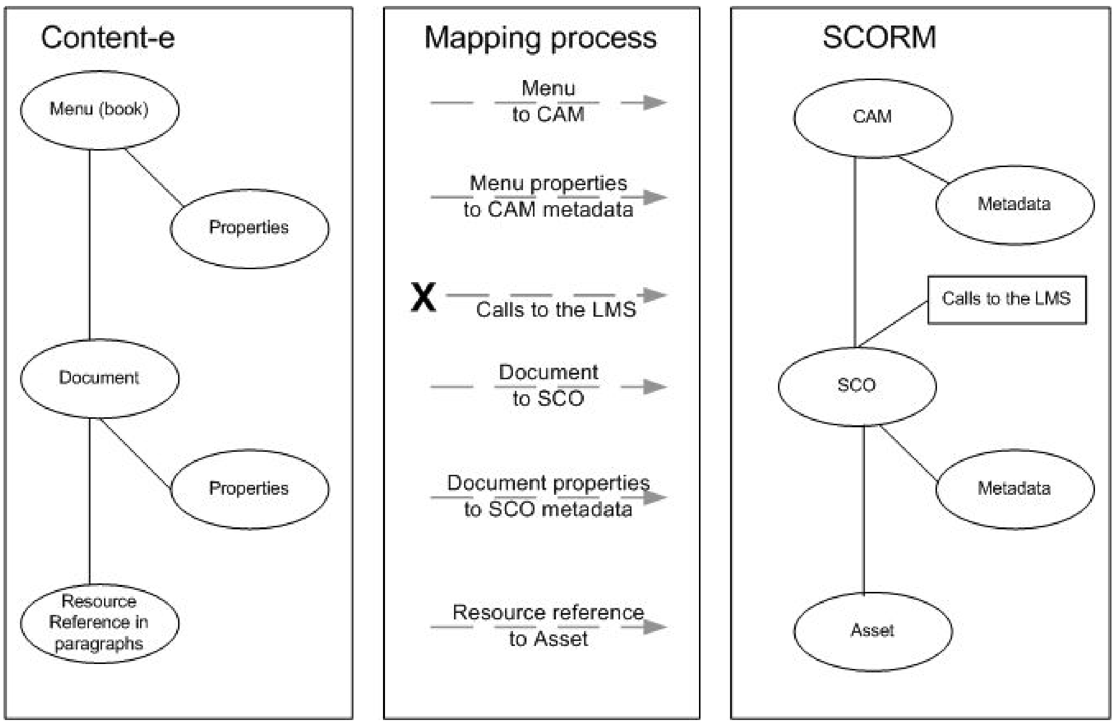


Figure 10: Visualization of the mapping process

In Figure 10 the mapping is defined for the data set in Table 2. In the mapping process in this figure the 'X' describes the fact that calls to the LMS can't be mapped from Content-e, but have to be generated in the mapping process.

3.3. Mapping matches between Content-e and SCORM

The mapping process can be defined on the matches found in the previous section. The process should answer the three remaining questions:

- How is the data retrieved from the Content-e database?
- How can this data be transformed to a SCORM conforming package?
- How should interaction with the LMS, in which the package will be placed, be implemented in the documents of the package?

In section 3.2.4. concerning the metadata a remark was made that a preparation stage has to be incorporated in the process of exporting data from Content-e to SCORM. This involves the creation of a specific property structure to be used for the SCORM publication. This creation needs to performed only once in Content-e, before the generation of the first SCORM publication. This allows authors to enter values for the properties to be transformed to metadata.

The mapping process involves the generation of HTML and XML files, which form a learning experience conforming to SCORM.
The HTML files are needed to implement the SCOs and will be shown in the LMS.
Two different types of XML files have to be generated:

- the manifest file of the learning experience, containing the logic and structure
- the metadata files, describing the CAM and SCOs

The metadata can be constructed as separate files or be put inline in the manifest. Both options are valid according to the SCORM standard. Separate files have preference, since this allows metadata to be validated more easily.
Methods for transforming the data housed in Content-e to the required files in a SCORM package are discussed after exploring the preparation stage on which the mapping process depends.

3.3.1. Preparation stage

The preparation stage is meant to construct a closely resembling property structure of the metadata structure in SCORM. This will alleviate the work involved in mapping properties to metadata.
Difficulties however occur, because SCORM metadata offer some capabilities, not found in the property functionality of Content-e.

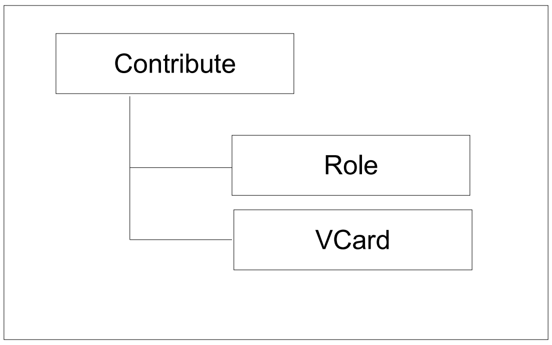
Three areas in which difficulties can occur:

1. Repetition of property structures
2. Limiting the number of repetitions
3. Changes in the prepared property structure

These areas are explored further in the following sections.

3.3.1.1.Difficulty of repetition of property structures

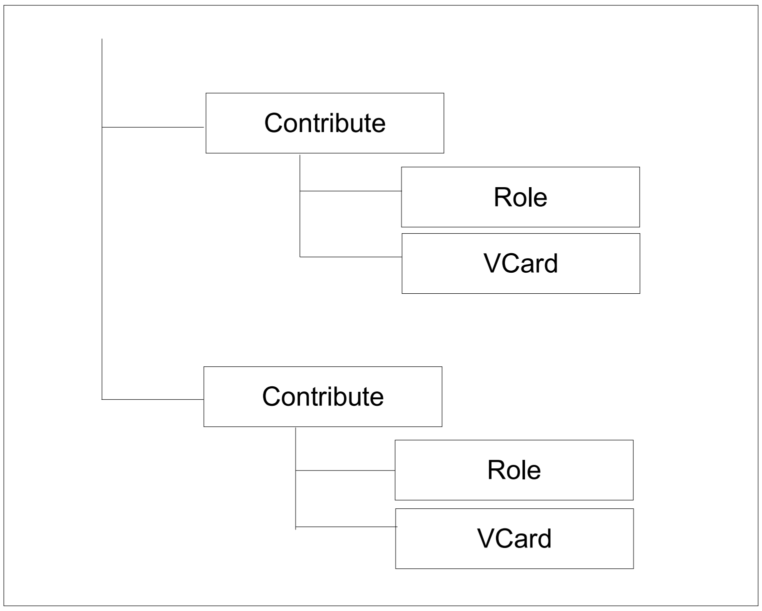
This will be explained by the following example.
Assume the following metadata structure in SCORM, as shown in Example 6:
An element 'contribute' has a child element 'role' and a child element 'vcard'.
This corresponds with the contribution a person or organization has made to the learning materials, and is explained by the role this person or organization has played and a electronic business card of him or it.



Example 6: The contribute element

This structure can also be created with properties in Content-e:
A property 'contribute' can have a child property 'role' and a child property 'vcard'.
The mapping from Content-e will be straightforward.

However, it is possible to specify in SCORM that 'contribute' occurs multiple times. The user is allowed to repeat this structure, with a possible upper limit on the number of times of the repetition.
The previous structure can be repeated multiple times:
An element 'contribute' has a child element 'role' and a child element 'vcard'. If the user needs to add another person or organization, he would like to add another 'contribute' element with again 'role' and 'vcard' as children.
Example 7 describes this situation.

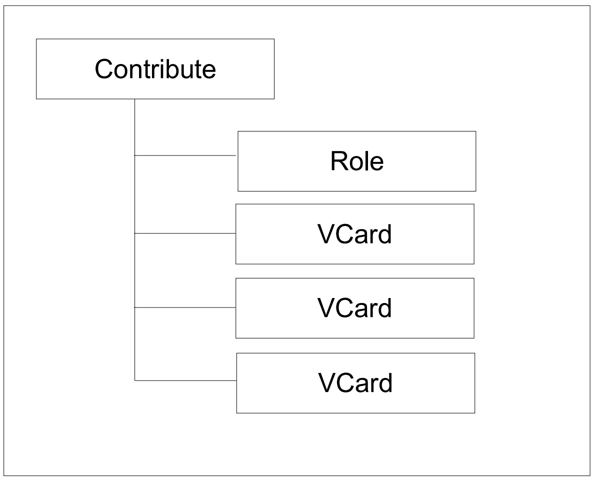


Example 7: Repetition of the contribute element

This structure can be created in Content-e. The user with an administrator role in Content-e needs to know in advance the maximum number of times a normal user would want to add the contribution element including child properties. He can then implement this property structure. The easiest way would be to set a fixed number of these structures, but this approach disallows an author to add more structures and acts inflexible.

3.3.1.2.Difficulty of limiting the number of repetitions

A new situation can be sketched, which continues with the previous example of the element 'contribute'. Instead of repeating a structure, a single element is repeated. In SCORM the restriction can be defined to repeat an element a maximum number of times.
In Example 8 a user has decided to repeat the 'vcard' element three times. This is valid if the 'vcard' element has the restriction that it can be repeated at most ten times.
This restriction is not built in the properties module of Content-e, a property can occur only once or an unlimited number of times, a limitation on the number of repetitions is not possible.



Example 8: Repetition of the vcard element

To continue with this example, Content-e can have fifteen 'vcard' elements.
The solution would be to leave the superfluous five elements out of the mapping process, leaving the author in a surprised state with the question why some elements were not mapped to SCORM.

3.3.1.3.Difficulty of changes in the prepared property structure

A conflict can occur as mentioned in [Sattler, 2003] with the use of the current property module. In this article conflicts and solutions to mapping processes of data from one database to another database are discussed. The situation of databases can be extended to structures like the property tree structure of Content-e and the metadata structure of SCORM.
The meta conflict, which is a structural conflict, is of special interest to the mapping process of properties to metadata.
Such a conflict occurs when a instance of data in one table is represented by a schema object in another table. This can be extended to the situation of mapping properties to metadata. In the property structure an instance of a property is

made with a name, while an element with the same name is a part of the structure of the metadata.

An example of this situation is shown in Table 3 and Table 4, the structures are represented by tables:

Contribute	***Role***	***VCard***
null	Author	FN: John Doe
null	Publisher	FN: Jane Doe

Table 3: Contribute according to a metadata structure

Table 3 contains two representations of the contribute element used in a metadata structure.

ID	***Name***	***Value***	***ParentID***
1	Contribute	null	null
2	Role	Author	1
3	VCard	FN: John Doe	1
4	Contribute	null	null
5	Role	Publisher	4
6	VCard	FN: Jane Doe	4

Table 4: Simplified properties table

Table 4 contains a simplified version of the property module used in Content-e, to clarify where the problem lies.
In Table 3 the elements 'contribute', 'role' and 'vcard' are part of the structure. In Table 4 those same elements are used as values in the rows. It could happen that 'Role' was changed to 'Role2' in the fifth row of this table. This would not break any structure, but cause a problem when trying to map values in the rows of Table 4 to columns in Table 3. 'Role2' is not part of the structure in Table 3 and mapping would fail due to this problem.

The situation sketched above occurs when the property structure meant to be mapped to metadata is tampered with by a user of Content-e. This user could decide to change the name of the child 'role' property of the second 'contribute' property into the name 'role2'. The change of name of a property can be performed in the 'Properties' tab sheet in the Management System of Content-e.

3.3.1.4.Summary preparation problems and their solutions

Summarizing the problems with properties in Content-e:
1. Dynamic repetition of structures is not possible.
2. An upper limit on the number of repetitions of an element is not possible.
3. Property structure can change.

These problems pose difficulties in allowing users to enter properties, which will be mapped to metadata. However some workarounds are available.

1. Workaround for the problem with dynamic repetitions:
   Restricting the amount of structures. This is done by statically creating the structures and involves a large amount of time to create them. For structures with no upper limit on the number of repetitions, a decision has to be made on creating a feasible upper limit.
2. Workaround for the problem with the upper limit:
   Limits for the properties, which are to be mapped to SCORM, can be derived from the SCORM standard. Again this involves a large amount of time.
   The upper limit problem can be circumvented in the transformation process by leaving out elements if needed, such that the upper limit is respected in SCORM. This implies possible loss of information.
3. Workaround for the problem with the change of the property structure:
   A specific set of properties are created for the sole purpose of mapping them to metadata conforming to SCORM. After creation they aren't to be changed by another user working with Content-e. This is a requirement to let the mapping of properties to metadata work.

All these proposed solutions are quite inflexible and a real solution needs to be created.

A new design has been made to implement the properties differently, trying to present a better solution for the problems mentioned above. This design uses a component in Content-e, the XML editor.
The XML editor has been designed to enable the use of XML in Content-e. It is the end result of research performed for a previous thesis [Poel, 2002].
The editor is dependent on two files:
- a XML Schema, to know which structures are possible as data input
- a XSLT, to display a form in which the data can be entered

At first this editor has been designed to create exercises where data is contained in XML format, and different forms have created to edit and to display the data. This provides a clean separation between display and data. Editing data can only be done if it is allowed by a XML Schema, thereby allowing producers of exercises to limit the number of structures and making validation of the data easier.
This editor also has other applications such as creating animations in Content-e.

This results in two approaches for constructing metadata:
1. use of the XML editor with properties
2. use of current property module of Content-e

In the remainder of this thesis both approaches for constructing metadata will be explored. If the approach with the XML editor turns out not to work, the other approach at least is available to be able to produce a SCORM conforming package.

3.3.1.5.Using the XML editor with properties

While dealing with problems of creating property structures in Content-e, the possibility of creating these structures with the use of the XML editor appeared. Since the metadata structures in SCORM are defined by a XML Schema, creation of those structures in this editor shouldn't be a problem. This also has the advantage that a transformation wouldn't be necessary anymore for transforming

properties to metadata, the properties are structured exactly the same as the metadata.
Generalising this approach to be able to create any property structure, not restricting it to be SCORM compatible, but in fact to be compatible to other standards as well led to a new design for creating and storing properties.
The only requirement is that for any of the structures to be produced a XML Schema needs to be available.
Some extra requirements have been established when using properties in combination with the XML editor:

- Groups of users with different permissions should only be able to edit data to which they should have access.
- Support for multiple languages. Since Content-e is multilingual, this functionality should be incorporated in the new property system as well.

The new property system should also be modular, as is the case with the current property module. In the design for the new properties a more relational approach has been taken to connect properties to 'menus', 'documents' and 'content'.

The idea is that a property structure, in XML format, conforms to a XML Schema. One or more forms correspond to the schema, each allowing to edit the values of a different part of the structure defined by the schema. Each form has its own corresponding language, captions and descriptions written in this language. A group in Content-e has access to a set of forms, which they can use, and one of these forms act as default when they work with the XML editor.

The following entities have been recognized to meet the requirements:

- Properties
- Schemas
- Translations
- Stylesheets
- Groups

Since the new property system has to be modular, Content-e should be able to run without the properties. Two systems are recognized, Content-e and the properties. On database level, the tables used in Content-e can also be separated from the tables needed for the property system.
However a layer has to be introduced to connect those two systems and in order to prevent the property system getting too deeply hooked into Content-e. This can be done by making sure that the tables of Content-e don't contain foreign keys to the tables involving the properties. The layer consists of tables, which connect to the tables of Content-e and of the properties.
This can also be modelled as three layers:

1. Content-e layer. This layer consists of the tables on which Content-e operates.
2. Connection Layer. This layer contains tables, which connect tables in the Property Module with tables in the Content-e layer.
3. Property Module layer.

These layers are visualized in Figure 11. The arrows represent dependencies. The connection layer depends on Content-e. The new property module and the connection layer depend on each other.

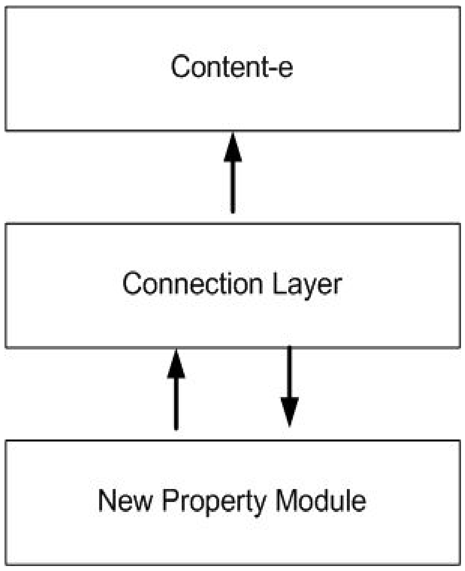


Figure 11: Representation of the layers in new property module approach

The need for defining a property structure in the Management System is gone, a XML Schema defines the structure. This solves the problem of a user being able to tamper with the structure.
The XML Schema for the SCORM metadata can be used in the XML editor, thereby dismissing the problems with repetitions and upper limits.

3.3.2. Main process of mapping

The questions which have to be addressed by the mapping are:

- How is the data retrieved from the Content-e database?
- How can this data be transformed to a SCORM conforming package?

HTML and XML files have to be generated, conforming with SCORM.

There are basically two approaches of performing the mapping:

- **Create the files directly from Content-e.**
  This approach implies a very short mapping process, and would be quite fast to implement. The drawback of this approach is the need for a rewrite if changes are made to structures in Content-e. It is also a highly customized solution, which doesn't allow reuse of files or structures generated by the mapping process.
- **Create an intermediate structure from Content-e.**
  The intermediate structure needs to contain sufficient information to create file conforming to the SCORM standard. The preferred format of this structure is XML. It is ideal to transform data to a different format, like HTML, but also to transform data to a different XML structure. These transformations can be performed with XSLT files.

Other publication formats could benefit from this intermediate structure as well. Writing custom XSLTs for these formats would result in the desired HTML or XML files for a different publication format. The data retrieved from the database is available in a clear structure and can be processed with other tools understanding XML.
Another advantage is that this structure could be used to import the content from one Content-e system to another.
These advantages require XML structures to be created also on higher and lower levels than a book. Basically a structure representing a menu at room, desk or book level has to be created as well as the ability to create a structure representing a document.

Considering these last mentioned advantages, the second approach is used in the mapping process.

In Figure 12 the mapping process with the intermediate XML structure is shown. The mapping process is actually divided in two processes:

1. Data is extracted from the database of Content-e and is stored in the XML intermediate structure. This will address a question of the main problem: "How is the data retrieved from the Content-e database?"
2. The other remaining question, "How can this data be transformed to a SCORM conforming package?", is addressed in the transformation process from the intermediate XML structure to files in SCORM format.

The first process defines how the XML structure is created and supplied with data from the database of Content-e. A XML Schema needs to be constructed to specify the XML structure.
After construction the structure is supplied with data. Data retrieval from the Content-e database is discussed in the following sections.

The second process uses this structure to produce files conforming to SCORM. The XML Schema for the intermediate XML structure is compared with the XML Schemas provided by ADL. The comparison results in the specification of the transformation of the data in the intermediate structure to data conforming to the XML Schemas.
The mappings as shown in Figure 10 are performed by this second process. Figure 12 provides a more technical view than Figure 10.

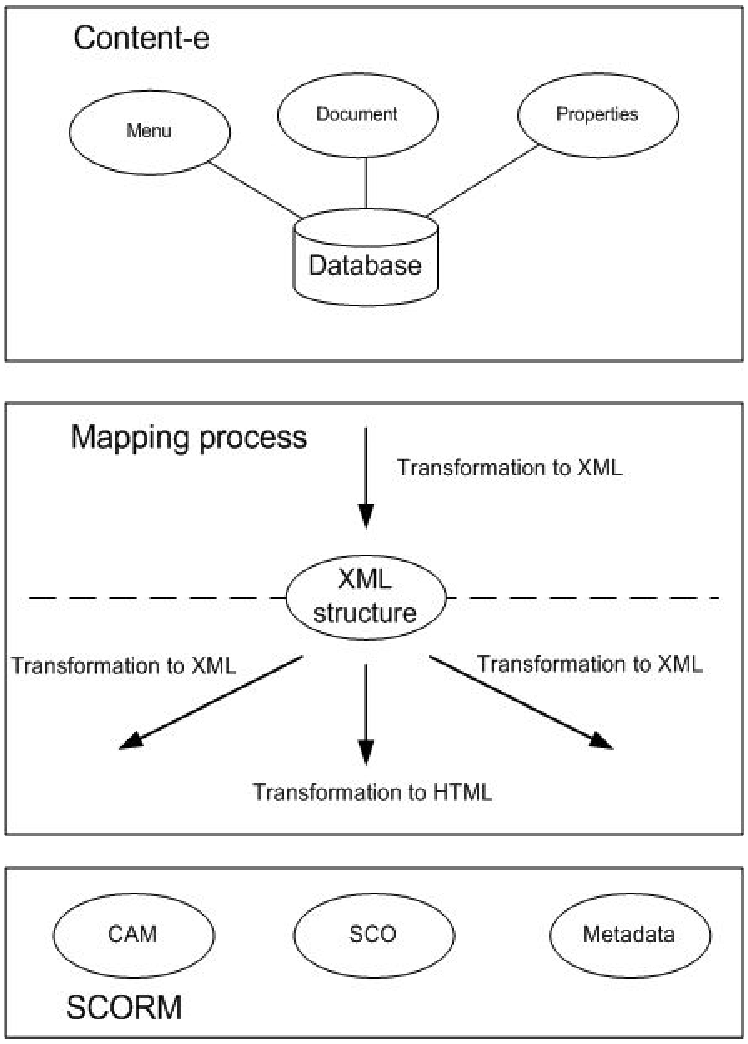


Figure 12: Mapping process using XML

3.3.2.1.Process of creating the intermediate XML structure

The process of creating the intermediate XML structure defines the XML structure and supplies it with data from the database of Content-e. Defining the structure is implemented with a XML Schema. The process is discussed in the following sections.

3.3.2.1.1.Constructing the schema for the intermediate XML structure

A XML structure is used in the mapping process and it is important to implement a XML schema to define the possible structures. Since the data for the intermediate structure is constructed from a database, a mapping can be devised from the schemas of the relational tables in the Content-e database to the XML Schema. This mapping of schemas is called schema extraction.

3.3.2.1.2.Translation of relational tables to XML schema

Before a mapping of relational table schemas to XML Schema can be defined, it is important to know which modes define how columns should be mapped. There exist two modes: element-oriented and attribute-oriented mode. With the first mode columns are mapped to elements, the second mode maps columns to attributes.

Several methods exist to map from a relational database to XML and have been explored in [Lee, 2001] and [Lee, 2002], they will be briefly be discussed here:

- Flat Translation (FT)
  FT is the most simple translation. It will map columns to elements or attributes, depending on the choice of the previously explained modes. It lacks however the utilization of repeating subelements. When the element-oriented approach is used, an order is introduced in the XML, which was not there in the relational model.
- Nesting-Based Translation (NeT)
  NeT is designed to overcome the problem with FT. NeT is based on nesting of columns. This algorithm is useful for decreasing data redundancy and obtaining a more intuitive schema by removing redundancies caused by multivalued dependencies and performing grouping on attributes. NeT however only looks at one table at a time, thereby overlooking the relationships between tables.
- Constraint-Based Translation (CoT)
  CoT has been developed to overcome the problem with NeT of not considering inter-table relationships. It uses Inclusion Dependencies of the relational schema. The focus is on foreign key constraints, the most straight forward dependency. CoT builds a graph of the inclusion dependencies. So it actually visualizes the relationships between tables, the foreign keys. Once the graph is constructed, the algorithm can start translating. Starting points of the graphs are called top nodes. By following a path, prescribed by the algorithm, from these top nodes, the schema is constructed which describes the structure.
  In short the relational schemas are extracted from the database and transformed into XML schemas.

The last method is the most appropriate for mapping the relational database from Content-e to XML. Especially the use of inclusion dependencies corresponds with the use of primary and foreign keys used in this database. CoT is therefore used to create a XML schema from the database.
The implementation of the CoT algorithm is performed in the next chapter.

3.3.2.1.3.Supplying the intermediate XML structure with data

Three options exist for the supply of data for the intermediate XML structure:

- Retrieve data with SQL from the Content-e database
- Retrieve data with a XML extension from the Content-e database
- Retrieve data with ASP objects from the Content-e database

3.3.2.1.4.Retrieve data with SQL from the Content-e database

The database of Content-e is a relational database, so using SQL will work quite well to get the data from the tables and process it. This option is however probably the least attractive, all SQL queries need to be built, and after this step XML should also be constructed. The other options provide either XML output with specified SQL input or retrieve the data into objects, ready to be manipulated or read. Functionality to be built in this option is already incorporated existing in the others. This option should therefore be disregarded to retrieve data from Content-e.

3.3.2.1.5.Retrieve data with a XML extension from the Content-e database

Two options have been explored to create XML from a relational database:

1. **Microsoft SQL Server 2000 XML extension**
   Microsoft SQL Server 2000, the database system on which Content-e runs, has a XML extension which can be used in a SQL SELECT-statement. A SELECT statement is used to retrieve data from a relational database. By formatting the data retrieval query with specially constructed unions and using special XML keywords, the database server is able to output XML instead of normal relational query results. Normally when queries are constructed the number of tables involved are known. With tree like structures present in Content-e, a calculation would have to be performed to know the number of aliases for a table which are used. This calculation can be put in a Stored Procedure.

2. **Xperanto**
   Xperanto is a middleware system that runs on top of any relational database system. In [Shanmugasundaram, 2001] techniques are mentioned to retrieve data from XML using XQuery, a query language for XML objects. In its standard XML view of a relational database it just output the contents of tables as XML, but this is not the preferred way as it doesn't show the relations between the tables. However it is possible to create a custom XML view outputting these relations in a proper way. This is done with the use of special language constructs to specify what needs to be retrieved and how it should be outputted.

3.3.2.1.6.Retrieve data with ASP objects from the Content-e database

Objects programmed in ASP for Content-e can be used to perform the transformation from the data in the database of Content-e to the intermediate XML structure. Functionality incorporated in these objects such as the verification whether menus and documents should be retrieved for the user performing the mapping can be utilized.
The output of this approach are menu, document, paragraph and property objects linked together.
All objects in the application get their state from the data contained in the database, or if new objects are created, their data is stored in the database. The state of an object is defined by the values of its variables, these values are the previously mentioned data.
The step after the creation of the objects is to create XML out of their state.

3.3.2.1.7.Chosen approach to data retrieval

Two options exist for the data retrieval from the Content-e database and creation of the intermediate XML structure:

- use a XML extension
- use ASP objects written for Content-e.

The second option is chosen, as these objects already provide necessary logic, such as respecting the permissions system of Content-e, including content from other documents. The main step in the data retrieval would be to map the state of the objects to XML.
Using existing logic outweighs the automated construction of XML based on custom queries.

The creation of the requested XML can be done by traversing down from the top menu object to its document and paragraph objects. For every object considered in the traversal, variables contained in the object are read and put into subelements or attributes as specified in the XML Schema developed for the intermediate XML structure. Properties are mapped to subelements of the considered object.

3.3.2.2.Transforming data from the XML structure to SCORM conforming files

The XML Schema developed for the intermediate XML structure can be compared with the resulting XML Schema for the manifest and the metadata. The data stored in the intermediate structure contains enough information to build both type of XML files.

In Figure 13 a mapping needs to be defined from the a 'menu' XML fragment to an 'organization' XML fragment in the manifest. XSLT is perfect to perform this mapping. The 'organization' XML fragment represents the CAM, shown in previous figures of the mapping process.

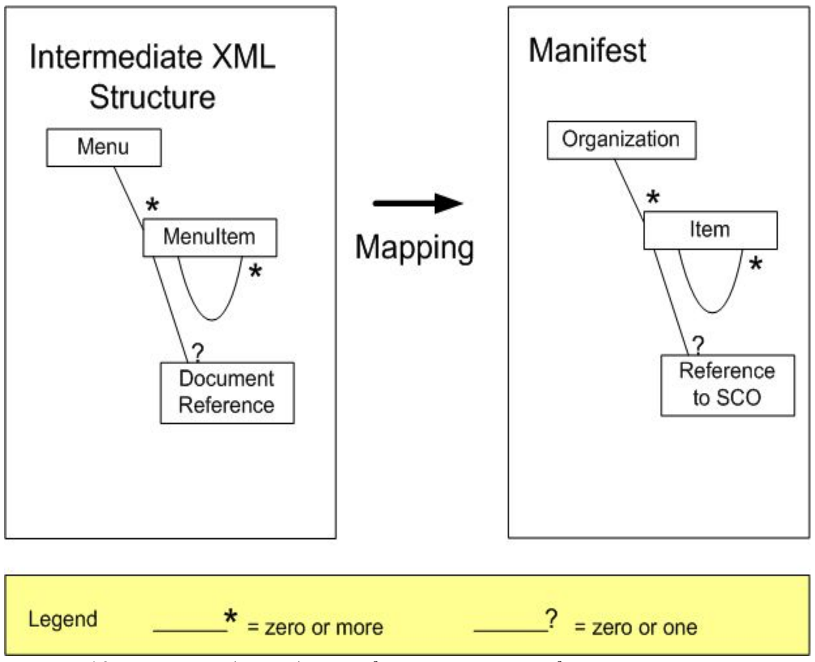


Figure 13: Mapping 'menu' XML fragment to manifest

In Figure 14 a mapping is needed between references to resources in the intermediate XML structure and dependencies of a resource in the manifest. A document in the XML structure is mapped to a resource in the manifest of type SCO.
If a resource in in the 'text' element of a 'content' element in the XML structure is referenced more than once, it is mapped to an asset. A dependency to this resource of type asset has to be created in the SCO.

If a resource in the structure is only referenced once, it can just be referenced by the SCO as a file. A link to this file needs to be created in the SCO.

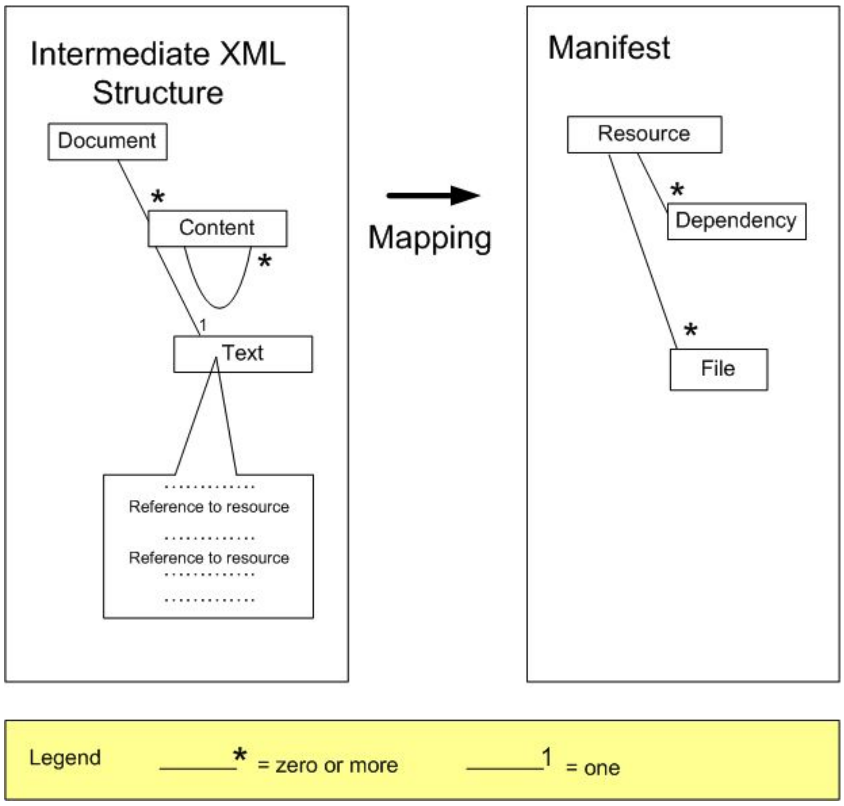


Figure 14: Mapping references in 'document' XML fragment to manifest

Two approaches were used to handle properties:

- using the current property module of Content-e
- using the XML editor

In the first approach 'property' elements in the XML structure are transformed to metadata.
This mapping is shown in Figure 15. The mapping is performed with XSLT, the input and output of the mapping are both in XML format.

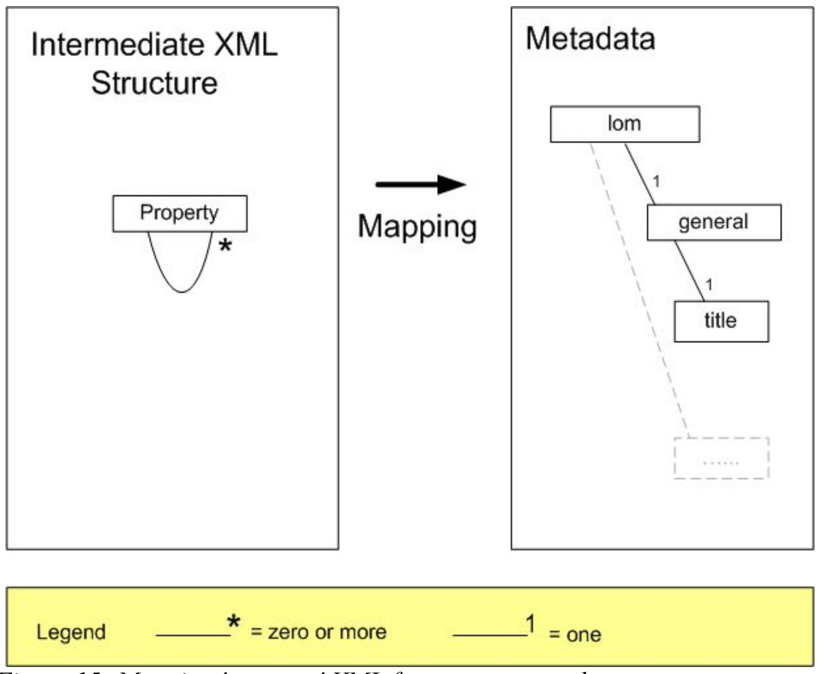


Figure 15: Mapping 'property' XML fragment to metadata

In the second approach a transformation is not necessary, because the properties are already in the required metadata format.

In addition to the generation of files, validating and testing whether the files are SCORM conformant can be performed.
The following validations and tests are available:
1. Validate the generated XML files with their XML Schema.
2. Test the generated SCORM pacakge with a test suite provided by ADL.
3. Load the SCORM package in a LMS supporting SCORM.

3.4. Main steps of matching

In Figure 16 the whole mapping process is visualized based on the decisions made in this chapter.

All questions of the main problem stated in chapter 1 have been addressed and decisions have been made. These decisions are used in the implementation phase discussed in the next chapter.

The main problem is stated again: How to export content in the underlying relational database of Content-e to a package of files conforming to the SCORM standard?

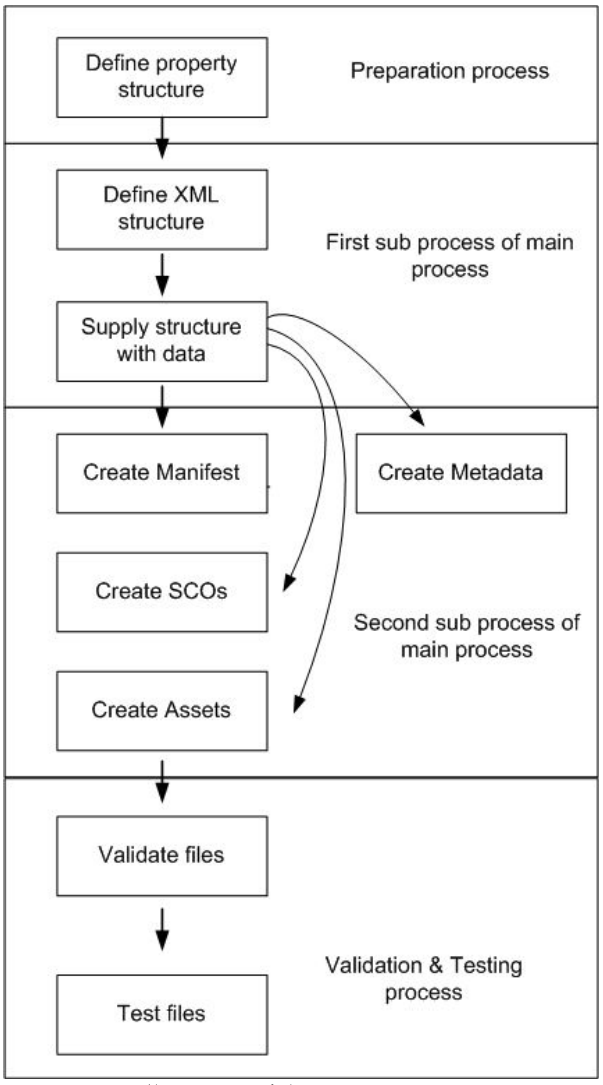

Define property structure
Preparation process
Define XML structure
First sub process of main process
Supply structure with data
Create Manifest
Create Metadata
Create SCOs
Second sub process of main process
Create Assets
Validate files
Validation & Testing process
Test files

Figure 16: All aspects of the mapping process

A summary of the questions, which together form the main problem and their answers are provided now:

1. At what level should the data be retrieved?

   Data is retrieved from book level.
2. How is the data retrieved from the Content-e database?
   ASP Objects programmed for the Content-e application are used.

3. How can this data be transformed to a SCORM conforming package?
   An intermediate XML structure is constructed from the data retrieved by the ASP objects. This structure is used to construct the files, which together form the SCORM package. A schema is constructed to define this XML structure.

Properties in Content-e pose some difficulties when transformed to metadata. These will have to be dealt with in a preparation stage.

4. How should interaction with the LMS, in which the package will be placed, be implemented in the documents of the package?
   Only signals for starting and stopping a SCO and an update of the status of a SCO are implemented as interaction. These will be implemented during the generation of a SCO. Adjustments to Content-e are not made.

4. Implementation of the mapping process

A plan is made to implement the decisions made in this chapter and summarized in the previous section, based on the answers provided in the previous section. The different steps are:

1. Preparation stage for the properties in Content-e.
2. Construction of the schema for the intermediate XML structure.
3. Storage of data in this structure using ASP objects.
4. Generation of metadata files from this structure.
5. Generation of SCOs, including required signals to be sent to the LMS, from this structure.
6. Generation of the manifest file from this structure.
7. Validation of the generated XML files for SCORM with XML Schema.
8. Testing the generated SCORM package.

Steps 1 and 2 only need to be performed once, to define structures.
Steps 3 to 7 are to be built in a SCORM publication of Content-e. The publication will let the user of Content-e select books to be published. This is the level at which data is exported, as concluded in the previous chapter.

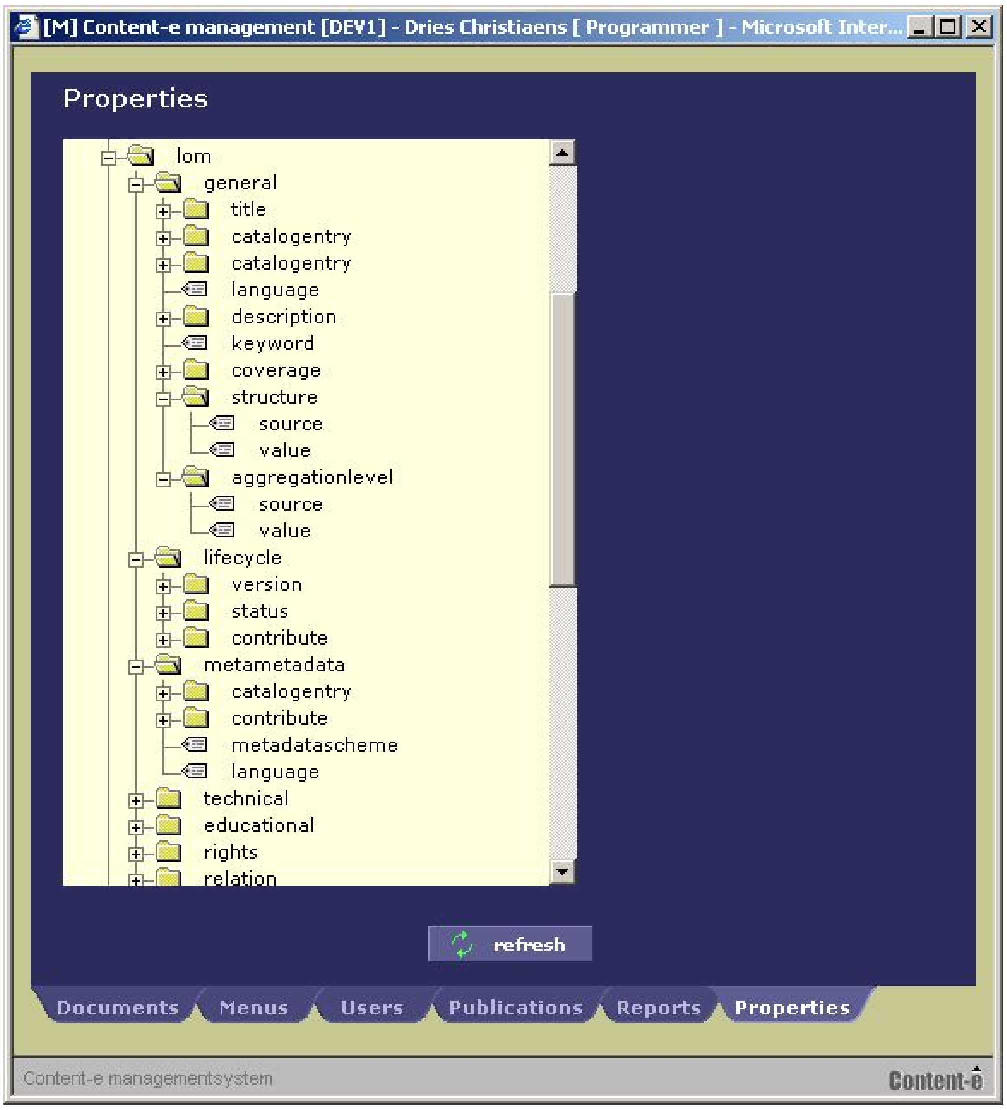
Every step of the plan is now discussed in the following sections.

4.1. Preparation stage for the properties in Content-e.

As stated in the previous chapter, two possible approaches have been devised to use properties: the use of the current property module and the use of the XML editor. First the use of the current property module is discussed.

4.1.1. Use of current property module

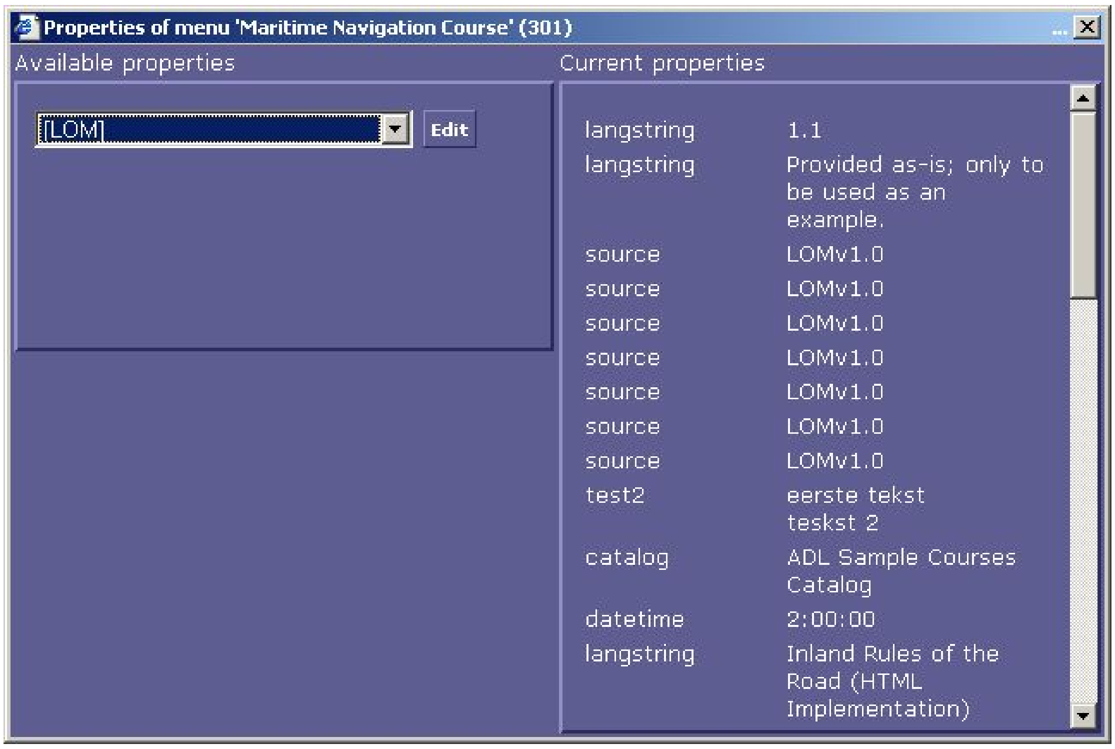
A tree structure has been built in the Management System of Content-e in the Properties tab sheet. It closely resembles the SCORM metadata structure.
In Figure 17 the property structure is partially shown to give an impression of the amount of properties, which have to created to allow the generation of metadata in the SCORM publication.
It is not in the interest of this thesis to explain what type every property has. The only remark to be made is that the property structure shouldn't be changed once it has been created.


[M] Content-e management [DEV1] - Dries Christiaens [Programmer] - Microsoft Inter...
Properties
refresh
Documents
Menus
Users
Publications
Reports
Properties
Content-e managementsystem
Content-ê

```
lom
  general
    title
    catalogentry
    catalogentry
    language
    description
    keyword
    coverage
    structure
      source
      value
    aggregationlevel
      source
      value
  lifecycle
    version
    status
    contribute
  metametadata
    catalogentry
    contribute
    metadatascheme
    language
  technical
  educational
  rights
  relation
```

Figure 17: SCORM property structure

Figure 18 shows values of properties entered in Content-e with the current property module. This screen also allows the update and removal of values for properties.



langstring	1.1
langstring	Provided as-is; only to be used as an example.
source	LOMv1.0
source	LOMv1.0
source	LOMv1.0
source	LOMv1.0
source	LOMv1.0
source	LOMv1.0
source	LOMv1.0
test2	eerste tekst teskst 2
catalog	ADL Sample Courses Catalog
datetime	2:00:00
langstring	Inland Rules of the Road (HTML Implementation)

Figure 18: properties, representations of SCORM metadata, in Content-e

4.1.2. Use of the XML editor

As already discussed, a design has been made to incorporate the use of the XML editor to add more complex properties to Content-e instead of using the current property system of Content-e. The end result will be a property structure corresponding exactly with the metadata structure defined in SCORM.
Originally the XML editor had the ability to validate if a node in a XML 'tree' was allowed to be attached to the 'tree' if a XML Schema was provided. This functionality has been removed from the XML editor, since it was not working with some XML Schemas. This removal however implies that the logic of adding elements to a XML 'tree' has to be provided in a different way to validate against the appropriate XML Schema.
A first approach is to implement a form for supplying values to properties, client side with the use of the XML Editor as a standalone application. The properties correspond with SCORM metadata. The output of the XML editor is a XML file and in this situation a metadata file.

4.1.2.1.Form

One of the most important files in the XML editor are the files responsible for the construction of the form displaying the data. These files are a XSLT file and a javascript file. Structures can be repeated, this information has to be made available to the form. This information is normally available in a XML Schema, however as previously mentioned the XML editor currently isn't able to use a XML Schema for this purpose. Also the SCORM standard specifies a maximum number of repetitions of structures, which haven't been implemented in the XML Schema as restrictions. So even if the XML Editor was supporting XML Schema while supplying data, further checks have to be performed.

4.1.2.2.Dependencies

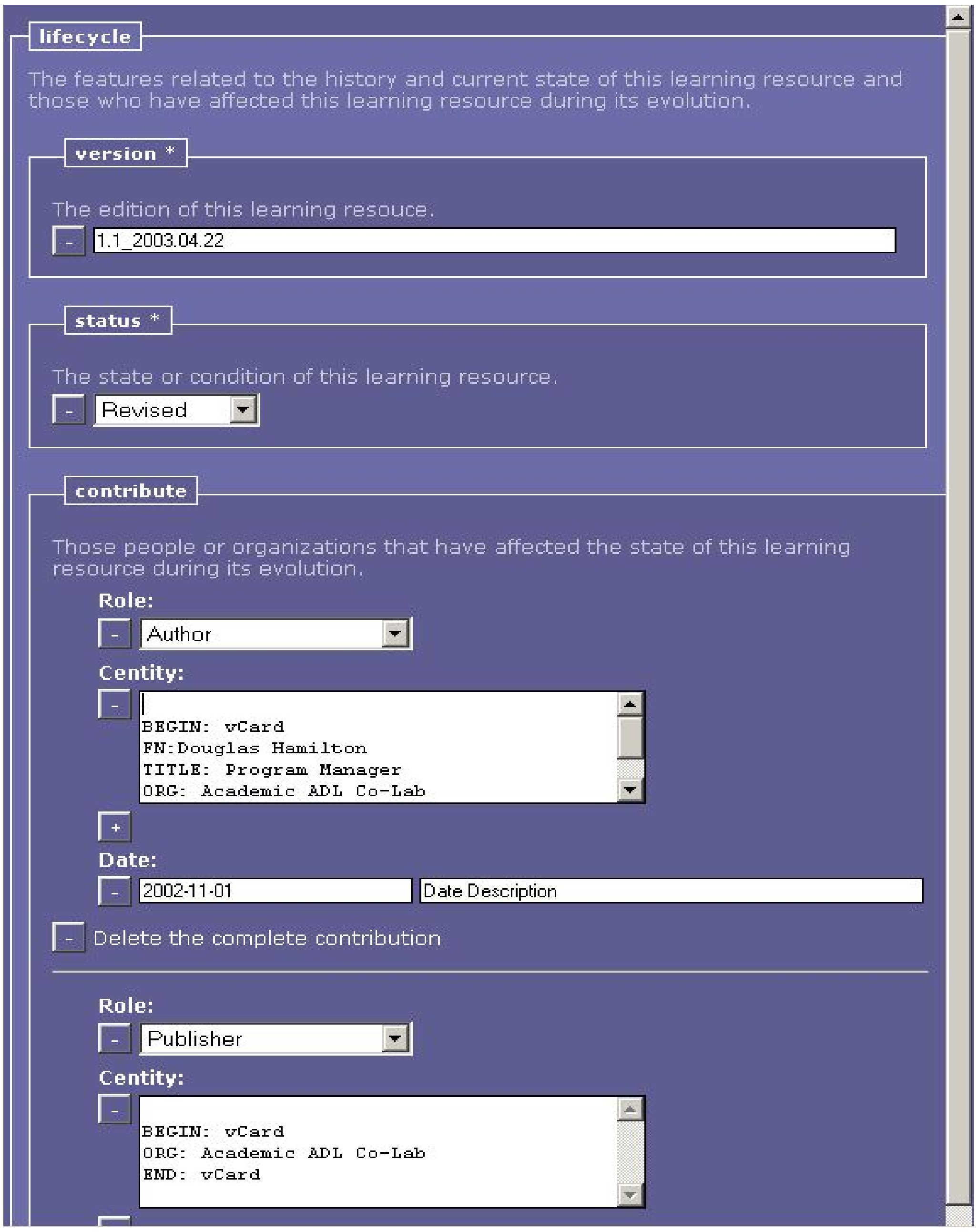
The XML Editor has several dependencies to provide a meaningful and functional form for supplying values to properties:

- **Language elements**
  In the form, values of elements have to be entered which represent languages. The format of these language elements have to conform to ISO 639[23] & ISO 3166[24] standards. However the representation of such a language element should be more readable. So a mapping is defined for language elements. In this first approach this mapping is implemented as a separate XML file.
- **Restrictions**
  As already mentioned restrictions have to be implemented to conform to SCORM. In the specification regarding the metadata information model, SCORM uses the term 'smallest permitted maximum', which indicates a maximum level to be implemented by every SCORM metadata supporting system. Such a system may not use a lower maximum. These restrictions have been implemented in a descriptions file.
  Another restriction is the 'vocabulary' type defined in the metadata information model. This type has two subtypes: 'source' and 'value'. For now all values are entered in the context of the LOM version 1.0 specification. The source therefore always has a predefined value of 'LOMv1.0'. Values of the 'value' type fall into two categories:
  1. Restricted, one value from a predefined set of options has to be chosen.
  2. Best Practice, options from a predefined set of options are available, however a value not in this set can also be chosen.
- **Description file**
  Every element of the SCORM metadata structure has its counterpart in a description file. Subelements of an element in the description file are:
  - mandatory (optional), the value of the attribute 'minlevel' indicates at what minimum level of granularity the element is mandatory.
  - maxoccurs, the number of times the element is allowed to occur.
  - element_description, the description belonging to the element.

  This file is of great importance in the construction of the form for supplying metadata. In it the structure is defined and it indicates for instance where new elements are to be added. Also the use of maxoccurs prevents people from supplying too much data, where it may not be allowed: the form will only allow addition of elements based on the value of maxoccurs.
  In essence the lack of validation with a XML Schema, has been overcome with the structure defined in the description file. Whenever the form has to be built, the information for all nodes can be retrieved from their counterpart in the description file, giving the appropriate user-interface to the person supplying the metadata.

4.1.2.3.Properties entered in the XML editor

In Figure 19 a screenshot is shown of the XML editor to give an impression how properties are handled. The result of the creation of the 'lifecycle' section in the form for SCORM metadata is displayed. Elements can be dynamically added through '+' buttons on the form. Each section of the form is also accompanied with a description to help the user of the XML editor. It will help in understanding what he is actually supplying data for. This is necessary since the number of values for metadata in SCORM can grow quite rapidly. Figure 19 provides a quite informative overview of data which has been entered by the user.



lifecycle

The features related to the history and current state of this learning resource and those who have affected this learning resource during its evolution.

version *

The edition of this learning resouce.

1.1_2003.04.22

status *

The state or condition of this learning resource.

Revised

contribute

Those people or organizations that have affected the state of this learning resource during its evolution.

Role:

Author

Centity:

```
BEGIN: vCard
FN:Douglas Hamilton
TITLE: Program Manager
ORG: Academic ADL Co-Lab
```

Date:

2002-11-01 | Date Description

Delete the complete contribution

Role:

Publisher

Centity:

```
BEGIN: vCard
ORG: Academic ADL Co-Lab
END: vCard
```

Figure 19: lifecycle form XML editor

Using a form with the XML Editor requires a person to build a XSLT file, which generates the form. In the case of the SCORM metadata this can get quite extensive, since a large number of different data elements can be added and be provided with values. The time to construct the XSLT can be a disadvantage, but this can outweigh the somewhat flexible form used in the current property module of Content-e.

4.1.2.4.Storage of the XML

The XML Editor delivers a XML file which should be stored, since people would want to save the data for other publications or internal use in Content-e. In the current design of creating properties with the XML Editor, the XML is stored as a large string in the database. This has some advantages and disadvantages.
Advantages:

- The XML is easily stored and retrieved from the database as a long string.
- Support for various XML Schemas is easy, since a different schema doesn't have the effect of having to create a number of tables, specifically needed to support that schema. It would just require another row in the table containing references to XML Schemas.
- A template is easy to build. It involves copying the XML string from one row to another, thereby relieving people from supplying a large amount of the same data multiple times.

Disadvantages:

- The amount of characters in a cell of a row in a table, supported by the database system can be limited. XML files larger than this limit would somehow have to be divided amongst a number of cells. At retrieval time the XML would also have to be reconstructed again.
- Searching inside the XML can pose a problem. SQL won't work, since the data isn't divided into columns. Every large character object would have to be loaded into memory into a XML object, on which a query with a XPath expression can be performed. For very large XML objects this would require a large amount of memory for the database server is resided, should the XML object be loaded by the database.
- Manipulating the XML can be only done by retrieving the complete XML object, perform an operation like changing the value of an element, and then storing the XML back into the database. This can be quite memory and time consuming for large XML objects.

All these disadvantages could have been solved by using a native XML database. Such a database would require a XML Schema, out of which it can construct tables, hidden for the user. Storing the XML in a XML native database involves sending the XML object or file with some query to the database and the database server will then deal with the storage. Querying the database can be done with a special query language named XQuery, designed to query XML files.
An example of a native XML database is Tamino.

However this would require a different type of database than currently used by Content-e, since Microsoft SQL Server 2000 is not a native XML database. It has some XML extensions built in, but this certainly doesn't qualify it as native XML database.

Another option is to create a mapping for every schema to relational tables. Various algorithms have been been developed to perform this mapping, such as can be seen in [Florescu, 1999]. The conclusion of this article of also storing the XML as a file on the filesystem due to the time involved for constructing XML out of relational tables is a bit disappointing.
The advantage of this mapping is the use of normal SQL, since relational tables are used to store the data. This offers possibilities to enable the search and retrieval of documents and menus within Content-e, based on some requested values for the metadata describing them, not just being able to export them to various formats. This resembles the functionality of the current property module in Content-e.
Two operations are needed to allow the mapping:

- assembly of elements into a full XML object/file
- shredding the XML object into various elements

Storage of XML in a relational database is not the main topic of this thesis and it deserves its own research.
The new capability of the XML editor to enter properties with more complex structures hasn't been integrated into Content-e, it has been developed as a prototype. A requirement of the integration with Content-e is the ability to use these properties in the search and filter functions in the Author and Management System.
An integration with Content-e is necessary to utilize the XML editor in the export of content from Content-e to SCORM.

4.2. Construction of the schema for the intermediate XML structure

A schema has to be constructed to describe the XML intermediate structure. In the previous chapter the CoT algorithm was chosen as best suited approach to generate this schema. The implementation of CoT is discussed here.
The schema will be created keeping in mind that properties have been provided in Content-e with a property structure, not with the XML editor.

The basic steps in CoT are to get an overview of the inclusion dependencies and to build a graph based on these dependencies. Travelling from a top node in the graph node to other nodes delivers a XSchema, a general schema for XML. A XSchema can be implemented as a XML Schema.

4.2.1. Preliminary work in the application of CoT

However before applying CoT preliminary work has to be performed. The first step, not mentioned in CoT, is to gather the relevant tables in the database, which contain the content. Tables and columns used for regulating the security in Content-e have no place here. They however are needed to retrieve the content, to which the current user has access, so they will not be put in the XML result, but are necessary to build the XML. CoT assumes that all data from tables in the database have to be transformed to XML, this is clearly not the case with Content-e, it would mean a breach in security, exposing all the data.
The relevant tables are:

- Menus
- MenuItems

- Documents
- Content
- ContentProperties
- PropertyNames
- PropertySelections
- PropertyOptions
- PropertyTypes

4.2.2. Creating the overview of the inclusion dependencies

The second step is to create the overview of inclusion dependencies (INDs).

Menus(me_id,me_name, me_description, me_startmessage, me_introdoc_id)
MenuItems(mi_id, mi_label, mi_document_id, mi_reference_id, mi_parent_id, me_id)
Documents(do_id, do_document_name, do_document_description)
Content(co_id, co_content, co_parent, co_url, do_id)
PropertyNames(pn_id, pn_name, pn_parent, pt_id)
PropertyTypes(pt_id, pt_name)
PropertySelections(po_id, ps_id, ps_value, ps_intvalue, ps_datevalue)
PropertyOptions(po_id, pn_id, po_code, po_option)	

Menus(me_introdoc_id)	$\subseteq$ Documents(do_id)
MenuItems(me_id)	$\subseteq$ Menus(me_id)
MenuItems(mi_reference_id)	$\subseteq$ Menus(me_id)
MenuItems(mi_document_id)	$\subseteq$ Documents(do_id)
MenuItems(mi_parent_id)	$\subseteq$ MenuItems(mi_id)
Content(do_id)	$\subseteq$ Documents(do_id)
Content(co_parent)	$\subseteq$ Content(co_id)
ContentProperties(pn_id)	$\subseteq$ PropertyNames(pn_id)
ContentProperties(ps_id)	$\subseteq$ PropertySelections(ps_id)
PropertySelections(po_id)	$\subseteq$ PropertyOptions(po_id)
PropertyOptions(pn_id)	$\subseteq$ PropertyNames(pn_id)
PropertyNames(pt_id)	$\subseteq$ PropertyTypes(pt_id)

Table 5: Schemas with INDs

It is important to know which columns are nullable, it will affect the creation of general schemas for XML, also called XSchemas. These schemas can be mapped to a XML Schema. Nullable means that the value in the column can contain a null value. They will be listed now:

- Menus(me_introdoc_id)
- MenuItems(mi_reference_id)
- MenuItems(mi_document_id)
- MenuItems(mi_parent_id)
- Content(co_parent)
- PropertySelections(po_id)

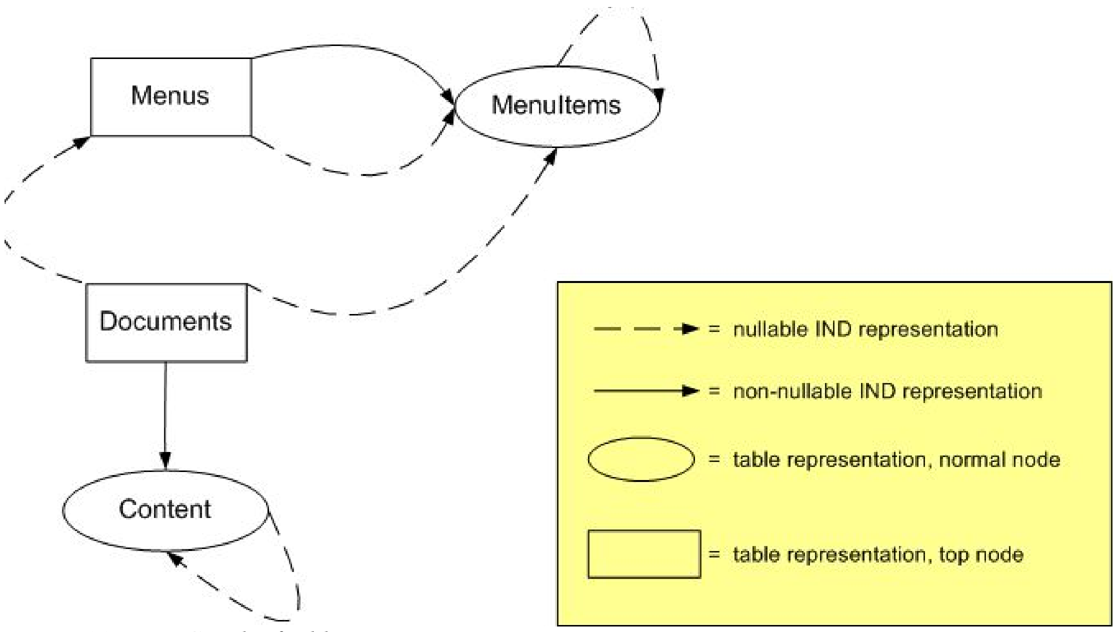

Menus
MenuItems
Documents
Content
= nullable IND representation
= non-nullable IND representation
= table representation, normal node
= table representation, top node

Figure 20: IND-Graph of tables representing content

4.2.3. Graph based on the inclusion dependencies

Out of these dependencies a graph can be devised, as shown in Figure 20 and Figure 21.
The creation of the IND-Graph displayed in Figure 20 is straightforward to make from the INDs given above. In [Lee, 2002] INDs with a nullable column are not mentioned in their IND-Graph, for completeness they are included in Figure 20.
Looking at the inclusion dependencies, the choice for top nodes will be 'Menus' and 'Documents'.
Thus far problems haven't been encountered in the application of the CoT translation. However in the next section the relational schemas of the properties are discussed, which pose some difficulties.

4.2.4. Difficulties in the relational schemas involving the properties

Tables concerning the properties pose a problem, as their connections to 'Menus', 'Documents' and 'Content' haven't be realised in a relational way. In order to provide flexibility an explicit relation in the database hasn't been made. Therefore schemas of tables concerning the properties can't be placed in the same IND-Graph of Figure 20.
In Figure 21 the IND-graph is shown for tables concerning properties. The table 'ContentProperties', of which a representation can be found in the figure, is used to connect to the three previously mentioned tables.
The tables 'PropertySelections' is used to hold the value of a property. It has four columns to represent the value, where only one column can actually contain the value of the property. This has be done to make comparisons in SQL statements easier. In a XSchema this advantage is gone and a type of a property can be retrieved from the 'PropertyTypes' table.
The decision therefore has been made not to use the last IND-graph to map to a Xschema, but to simplify the table schemas of the properties:

Properties(id, name, value, type, parent).
The 'id' is taken from the 'PropertyNames' table, as well as 'name', 'type' and 'parent'. The value of 'value' is taken from either the 'PropertySelections' table, or the 'PropertyOptions' table, based on where the value is not 'null'. The datatype of 'value' is changed to string, as this is a type in which all values can be represented.

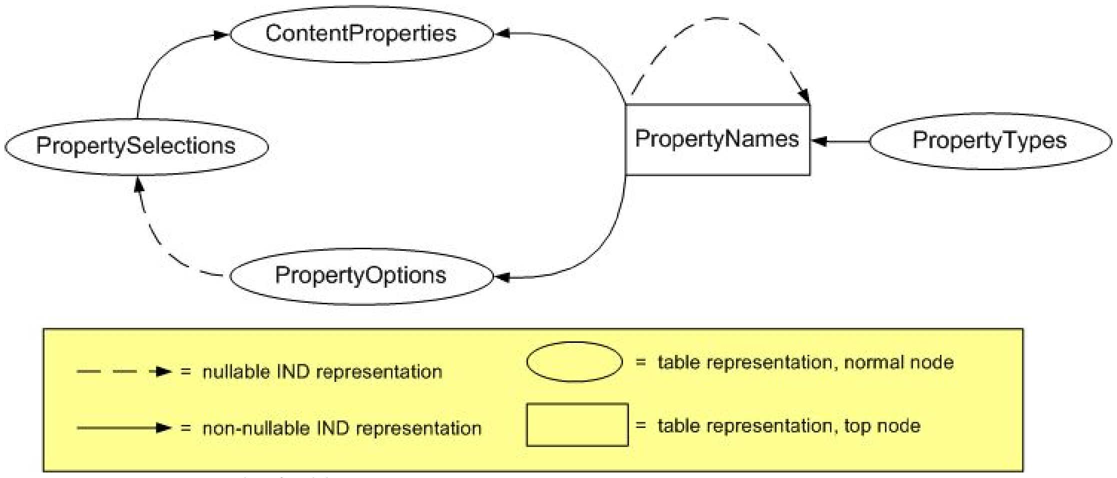

ContentProperties
PropertySelections
PropertyNames
PropertyTypes
PropertyOptions
= nullable IND representation
= non-nullable IND representation
= table representation, normal node
= table representation, top node

Figure 21: IND-Graph of tables representing properties

This approach simplifies the mapping to a XSchema for the properties considerably.

4.2.5. CoT and recursion

The examples mentioned in [Lee, 2002] don't discuss the possibility of recursion based on a entity referring to itself, like a parent-child relation, which can occur often in a database. To continue with the last figure, a instance of a 'PropertyNames' can have a parent, which is also of type 'PropertyNames'. This can be implemented with IDREFs, but if the requirement is met that a parent of a child can't also appear as its descendant, it is also allowed to map the child 'PropertyNames' as subelements instead of IDREFs. An endless loop in the recursion can't exist due to the requirement.
The inclusion dependency **MenuItems(mi_parent_id) $\subseteq$ MenuItems(mi_id)** will result in ***M*(MenuItems) = (mi_id, .. , .. , MenuItems*)**.
Parent-child relations also occur in the tables 'Content' and 'PropertyNames'.

4.2.6. Refinements in the schema for the intermediate structure

Also refinements can be made in the XSchema:
The 'co_url' column existing in the 'Content' table actually contains multiple values, mapping this to a single element in XSchema is not logical, it would be wiser to map this to multiple subelements. An alternative to a 'co_url' element would therefore be to create a 'excursion' element, which has an 'url' and a 'title' child element.
The plural form of the different entities, used in the relational schemas, doesn't seem natural to maintain in the XSchema. 'Documents', 'Menus', 'MenuItems' etc. are put in the XSchema as their singular form.

The constructed XSchema and XML Schema can be found in Appendix A.

In the following sections ASP, XML and XSLT are used to generate intermediate or end results. ASP can utilize XML and XSLT with the XML parser provided by Microsoft. Version 4 with service pack 2 of this parser is necessary, previous versions of the parser are unable to handle XML Schema correctly.

4.3. *Storage of data in the intermediate structure using ASP objects*

Now the schema for the intermediate structure has been deduced from the Content-e database, it is time to describe how this structure will be filled with data. As specified by the schema constructed in the previous section, the structure incorporates menus, documents, paragraphs and properties. The structure has to consist of enough data to generate files to be included in a SCORM package. Therefore data has to be retrieved at book level, corresponding to an aggregation in SCORM, and from all documents referenced by the chosen book. However in the implementation of the intermediate structure there isn't a restriction on the type of menu. A desk or room could be chosen as starting point of the data retrieval. This decision is in accordance with the schema, allowing menus to be referenced by other menus. It also creates possibilities to utilize this intermediate structure for other publication forms, which require data to be exported at room or desk level.

A class XMLWriter has been designed to implement to retrieve the data from the Content-e database and generate the XML structure. A UML diagram of this class is available in Appendix B.
Basically there are four methods of interest:

- **`XMLWriter.setMenu()`**
- **`XMLWriter.setDocument()`**
- **`XMLWriter.loadStructure()`**
- **`XMLWriter.getXMLObject()`**

There are two options for setting the level at which data retrieval should start:

1. A menu id has been set with **`setMenu()`**. All data should be loaded from the menu referenced by the menu id. This includes menu items used in the menu, the menus and documents referenced by them, as well as the paragraphs in these documents. Properties defined on these menus, documents and paragraphs are loaded as well.
2. A document id has been set with **`setDocument()`**. Data from the document and its paragraphs have to be loaded. Again it is necessary to load the properties of the document and the paragraphs.

A XML object *XMLOBJ* is created to represent the intermediate XML structure. Every menu, menu item, document, paragraph and property object retrieves their data from the database with SQL during their creation. After creation the object is mapped to a XML fragment and attached to *XMLOBJ*. The mapping involves creating a XML element with the name of a variable and its value for every variable in the object.

When the `loadStructure()` is finished, the XMLOBJ can be outputted to file or as an object, to allow further processing with ASP objects.

4.4. Generation of metadata files from this structure

Every menu, document and paragraph can have properties and they can have been attached as their children in the intermediate XML structure.
To construct metadata out of these properties a XSLT file has been written, to transform the properties, in XML format, into metadata according to SCORM. An example of the input of the transformation process is given in Example 9, the end result of this process is given in Example 10.

```
<document id="20">
  <properties>
    <property id="10">
      <name>general</name>
      <property id="11">
        <name>title</name>
        <property id="12">
          <name>langstring</name>
          <value>
            Conduct of Vessels in any Condition of Visibility
          </value>
        </property>
      </property>
      <property id="13">
        <name>language</name>
        <value>en-US</value>
      </property>
      <property id="14">
        <name>description</name>
        <property id="15">
          <name>langstring</name>
          <value>
            Discusses general rules of operation for vessels on
            inland waters. Topics discussed include: Look-out,
            Safe Speed, Collision, Channels, Traffic Separation.
          </value>
        </property>
      </property>
      <property id="16">
        <name>keyword</name>
        <property id="17">
          <name>langstring</name>
          <value>Vessel</value>
        </property>
      </property>
    </property>
  </properties>
</document>
```

Example 9: Snippet of an example of the intermediate XML file

The transformation basically converts the value of a 'name' element, e.g. 'general' into a new element with name 'general'. Child properties are converted into child elements. Elements with a value, e.g. properties with name 'langstring' in Example 9, are transformed into an element with a value. The property with name 'keyword' and its children are transformed into a 'keyword' element, containing a 'langstring' child element with value 'Vessel' which can be seen in Example 10.

```
<general>
  <title>
    <langstring>
      Conduct of Vessels in any Condition of Visibility
    </langstring>
  </title>
  <language>en-US</language>
  <description>
    <langstring>
      Discusses general rules of operation for vessels on
      inland waters. Topics discussed include: Look-out,
      Safe Speed, Collision, Channels, Traffic Separation.
    </langstring>
  </description>
  <keyword>
    <langstring>
      Vessel
    </langstring>
  </keyword>
</general>
```

Example 10: Snippet of an example of the resulting metadata file

The level of a SCO is set on a document, as already discussed in an earlier chapter. The structure of the menus in the intermediate XML format can now be mapped to a subtree in the manifest with the use of a XSLT. Properties in the intermediate format are mapped to separate metadata files.
A short decription of the class written in ASP to actually create the metadata files can be found in Appendix C.

A class MetadataWriter has been designed to generate metadata files. A UML diagram of this class is available in Appendix C.
Basically there one method of interest:
- **`MetadataWriter.generateMetadata()`**

The MetadataWriter class accepts an object representing the XML intermediate structure. It will use XSLT in combination with ASP to generate the metadata files in the **`generateMetadata()`** method.

4.5. Generation of SCO

In the intermediate XML structure the different documents are available as children of the 'documents' node. A 'document' node contains 'content' nodes, which represent the paragraphs of that document. Every 'content' node contains a 'text' node, containing the contents of a paragraph in HTML format. This is also the format in which the content was stored in the database. By supplying enough information for a content node, such as information about style, a XSLT file is able to construct a document in HTML format with the paragraph HTML portions in it shown properly.

An example of input of the mapping is shown in Example 11.

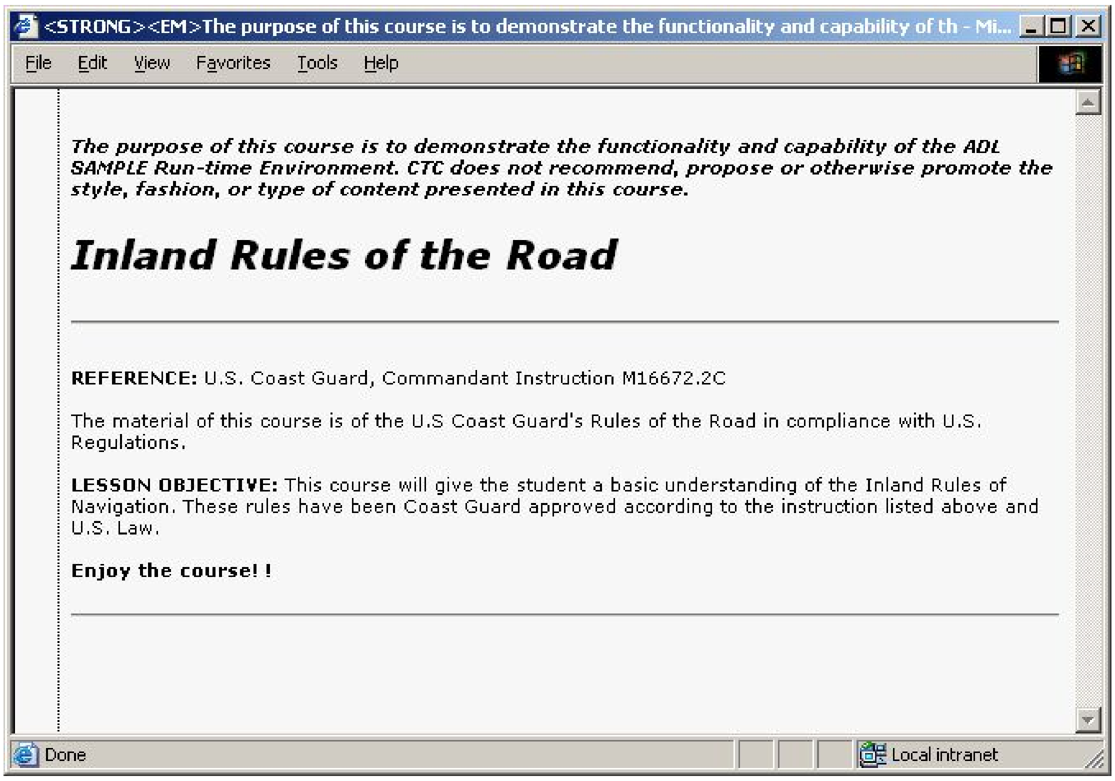
Since a SCO makes calls to the LMS, and a document is transformed into a SCO, the process of SCO creation should incorporate placing calls to the LMS inside the SCO.

All calls between **`LMSInitialize()`** and **`LMSFinish()`** to the LMS are valid and enable the transfer of data from and to the LMS.
As mentioned in the previous chapter a call will be implemented, setting the variable **`cmi.core.lesson_status`**. This variable can be set for every SCO, and obviously sets the status of the SCO to some predefined value. Initially the variable has the value 'not attempted'. For SCOs with only text in them, so without exercises, the predefined value 'completed' seems quite good as value to be set by a SCO just before a new SCO is loaded. This will let any teacher know which SCOs have been visited by a student.
A SCO has been designated as a document, a single HTML page. The method **`LMSInitialize()`** is called on the 'onload' event provided by Javascript in this page.
Setting the 'lesson_status' variable can be done while unloading the SCO in the 'onunload' event in the HTML page, after which the method **`LMSFinish()`** is called.

```
<document id="12804">
  <content id="46369">
    <style="1"/>
    <text>
      <![CDATA[ <STRONG><EM>The purpose of this course is to
       demonstrate the functionality and capability of the ADL SAMPLE
       Run-time Environment. CTC does not recommend,  propose or
       otherwise promote the style, fashion, or type of content
       presented in this course.</EM></STRONG>
      ]]>
    </text>
    <content id="46370">
       <style="19"/>
       <text>
         <![CDATA[ Inland Rules of the Road  ]]>
      </text>
    </content>
    <content id="46371">
       <style="19"/>
       <text><![CDATA[ <hr>  ]]></text>
       <content id="46372">
         <style="1"/>
         <text>
           <![CDATA[ <STRONG>REFERENCE:</STRONG> U.S. Coast
              Guard, Commandant Instruction M16672.2C
           ]]>
         </text>
       </content>
       <content id="46373">
         <style="1"/>
         <text>
           <![CDATA[ The material of this course is of the U.S
              Coast Guard's Rules of the Road in compliance
              with U.S. Regulations.
           ]]>
        </text>
       </content>
       <content id="46374">
         <style="1"/>
         <text>
           <![CDATA[ <STRONG>LESSON OBJECTIVE:</STRONG> This
              course will give the student a basic understanding of the
              Inland Rules of Navigation. These rules have been Coast
              Guard  approved according to the instruction listed
              above and U.S. Law.
           ]]>
         </text>
      </content>
    </content>
  </content>
</document>
```

Example 11: A document in the intermediate XML file.

Output of the mapping process loaded in the browser is shown in Figure 22. This mapping process is implemented in a file called 'documentGenerator.asp', available in all publications in Content-e. Functions in this file are applied to all documents to be published.



<STRONG><EM>The purpose of this course is to demonstrate the functionality and capability of th - Mi...

The purpose of this course is to demonstrate the functionality and capability of the ADL SAMPLE Run-time Environment. CTC does not recommend, propose or otherwise promote the style, fashion, or type of content presented in this course.

Inland Rules of the Road

REFERENCE: U.S. Coast Guard, Commandant Instruction M16672.2C

The material of this course is of the U.S Coast Guard's Rules of the Road in compliance with U.S. Regulations.

LESSON OBJECTIVE: This course will give the student a basic understanding of the Inland Rules of Navigation. These rules have been Coast Guard approved according to the instruction listed above and U.S. Law.

Enjoy the course! !

Figure 22: End result of mapping to a SCO

4.6. Generation of the manifest file

The section 'menu' in the intermediate XML structure is a XML representation of a menu structure found in Content-e. In Example 12 this section is shown for the book 'Maritime Navigation Course'. A menu node has a name and child menuitem nodes. A menuitem node contains a label and can contain child menuitem nodes. It can also contain a reference to a document.

The manifest consists of several subsections. The section 'menus' in the intermediate XML structure is transformed to the section 'organizations' in the manifest. The element 'Organizations' needs a default 'organization' to let the LMS know, when it imports the manifest, what structure to present to the user of the LMS. The organization node contains a name and child item nodes. Each item node has a title and can reference a SCO. It can also contain child item nodes.

```
<menu id="301">
  <type="3">
  <name>Maritime Navigation Course</name>
    <menu_item id="1836">
      <label>Maritime Navigation</label>
      <menu_item id="1837">
        <label>Inland Rules of the Road - Introduction</label>
          <menu_item id="9541">
            <label>Introduction</label>
            <document_reference>12804</document_reference>
          </menu_item>
        <menu_item id="1838">
          <label>Steering and Sailing Rules 1</label>
          <document_reference>4102</document_reference>
        </menu_item>
      <menu_item id="9540">
        <label>Steering and Sailing Rules 2</label>
        <document_reference>12803</document_reference>
      </menu_item>
      <menu_item id="1840">
        <label>Steering and Sailing Rules 3</label>
        <document_reference>4104</document_reference>
      </menu_item>
    </menu_item>
    <menu_item id="1830">
      <label>Lights and Shapes</label>
      <document_reference>4105</document_reference>
    </menu_item>
    <menu_item id="1831">
      <label>Sound and Light Signals</label>
      <document_reference>4106</document_reference>
    </menu_item>
    <menu_item id="1832">
      <label>Exam</label>
      <document_reference>4107</document_reference>
    </menu_item>
  </menu_item>
</menu>
```

Example 12: Snippet of a section of intermediate XML file containing the 'menu' element

```
<organizations default="organization1">
  <organization identifier="organization1">
    <title>Maritime Navigation Course</title>
    <item identifier="item_1836">
      <title>Maritime Navigation</title>
      <item identifier="item_1837">
        <title>Inland Rules of the Road - Introduction</title>
        <item identifier="item_9541" identifierref="sco12804">
          <title>Introduction</title>
        </item>
        <item identifier="item_1838" identifierref="sco4102">
          <title>Steering and Sailing Rules 1</title>
        </item>
        <item identifier="item_9540" identifierref="sco12803">
          <title>Steering and Sailing Rules 2</title>
        </item>
        <item identifier="item_1840" identifierref="sco4104">
          <title>Steering and Sailing Rules 3</title>
        </item>
      </item>
      <item identifier="item_1830" identifierref="sco4105">
        <title>Lights and Shapes</title>
      </item>
      <item identifier="item_1831" identifierref="sco4106">
        <title>Sound and Light Signals</title>
      </item>
      <item identifier="item_1832" identifierref="sco4107">
        <title>Exam</title>
      </item>
    </item>
  </organization>
</organizations>
```

Example 13: Snippet of a section of manifest containing the organizational structure

The XSLT performing this operation is shown in Example 14.
Futhermore SCOs and assets are created as resources in the manifest. A UML diagram is shown in Appendix C for the ASP class developed to create the complete manifest. The method of interest is the **`generateManifest()`** method of the ManifestWriter class.

```
<xsl:template match="/menu">
  <xsl:element name="organizations">
    <xsl:attribute name="default">
      organization1
    </xsl:attribute>
    <xsl:element name="organization">
      <xsl:attribute name="identifier">
         organization1
      </xsl:attribute>
      <xsl:element name="title">
        <xsl:value-of select="./name"/>
      </xsl:element>
      <xsl:apply-templates select="menu_item" />
    </xsl:element>
  </xsl:element>
</xsl:template>

<xsl:template match="menu_item">
  <xsl:element name="item">
    <xsl:attribute name="identifier">
      <xsl:value-of select="concat('item_',@id)"/>
    </xsl:attribute>
    <xsl:if test="./document_reference">
      <xsl:attribute name="identifierref">
        <xsl:value-of select="concat
('sco',./document_reference)"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:element name="title">
      <xsl:value-of select="./label"/>
    </xsl:element>
    <xsl:apply-templates select="menu_item" />
  </xsl:element>
</xsl:template>
```

Example 14: The XSLT needed to perform the transformation of menu element to organization section in the manifest.

4.7. Validation of the generated XML files for SCORM with XML Schema.

ADL has provided schemas to which the manifest and metadata have to conform. As already explained XML Schemas restrict possible XML structures.

4.7.1. Validation of the metadata files

The schema for the metadata is quite flexible, but this has the disadvantage that validation with XML validators is problematic. In the next section the problems encountered with the XML Schema for the metadata are discussed as well as their solutions.

4.7.1.1.Problems and solutions with the use of the schema for the metadata

The problems are:

1. Declaration of the XML namespace in the schema. This namespace is however reserved, thereby causing commercial XML validators to complain. It should not be declared in a XML Schema.
2. The schema allows extra elements to be added, which have to comply with their own schema. XML Validators however stumble over this option. They can't determine if elements conform to the metadata schema, or are part of other schemas.

```
1.<xsd:schema
2.targetNamespace="http://www.imsglobal.org/xsd/imsmd_rootv1p2p1"
3.            xmlns:xml="http://www.w3.org/XML/1998/namespace"
4.            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5.            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6.            xmlns="http://www.imsglobal.org/xsd/imsmd_rootv1p2p1"
7.            elementFormDefault="qualified"
8.            version="1.2:1.1 IMS:MD1.2">
```

Example 15: Piece of the metadata schema with namespace declaration

In Example 15 a part of the metadata schema from SCORM is shown and line three in this example is causing problems with XML validators.
The decision has been made to create a derived version of the metadata schema, where the first problem with the namespace is solved by removing the declaration of this namespace from the schema (line three from the example has been deleted).

The second problem can be seen in Example 16. On line twelve a reference is made to a element group 'grp.any'. This is just a example of a reference to this group, more references exist in the schema. These references however are causing non-deterministic behaviour in the schema, at least this is the complaint made by various XML validators. As the focus is on constructing metadata conforming to SCORM only and not to also other schemas, these references are not needed.
The approach for the solution of the first problem is also applied in the solution for the second problem. References to 'grp.any' are removed from the schema, thus removing the non-deterministic elements in the schema.

The result of these removal operations are a stricter schema, which allow validation by XML validators. This schema will be used in the validation of the metadata files produced by the transformation process.

```
<xsd:complexType name="generalType" mixed="true">
  <xsd:sequence>
     <xsd:element ref="identifier" minOccurs="0"/>
     <xsd:element ref="title" minOccurs="0"/>
     <xsd:element ref="catalogentry" minOccurs="0"
maxOccurs="unbounded"/>
     <xsd:element ref="language" minOccurs="0"
maxOccurs="unbounded"/>
     <xsd:element ref="description" minOccurs="0"
maxOccurs="unbounded"/>
     <xsd:element ref="keyword" minOccurs="0"
maxOccurs="unbounded"/>
     <xsd:element ref="coverage" minOccurs="0"
maxOccurs="unbounded"/>
     <xsd:element ref="structure" minOccurs="0"/>
     <xsd:element ref="aggregationlevel" minOccurs="0"/>
     <xsd:group ref="grp.any"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:group name="grp.any">
   <xsd:annotation>
      <xsd:documentation>Any namespaced element from any
namespace may be used for an &quot;any&quot; element.  The
namespace for the imported element must be defined in the instance,
and the schema must be imported.
   </xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:any namespace="##any" processContents="strict"
minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:group>
```

Example 16: Piece of the metadata schema allowing extra elements conforming to other schemas

4.7.1.2.Mandatory elements in the metadata specification

The schema for the metadata should basically be an implementation of the specification for the metadata. In the specification of SCORM it is stated that some elements are mandatory. Actually there are two types of mandatory elements, which can be distinguished:

1. Mandatory type A, this type is always mandatory, not depending on the granularity of the object which is described.
2. Mandatory type B, this type is mandatory, depending on the granularity of the object which is described. An element can be for instance mandatory if it describing a SCO or CAM, but is optional if an asset is described.

The schema written by IMS and provided by ADL however only contains optional values, which is somewhat strange. In the metadata specification there exist elements of Mandatory Type A, and this could have been implemented in a XML Schema as there are attributes, which can enforce elements to exist in the XML file conforming to the XML Schema.

However the choice of making elements required has been made by ADL to incorporate in the SCORM standard, not by IMS. Other standards can make the choice of requiring other elements.

Since metadata has to be checked for both Mandatory Type A and B, only validating the metadata against the provided XML Schema is not enough to let it conform to the SCORM standard.
So besides validating metadata against a XML Schema, validation has to be performed for the Mandatory Type A and B elements.
In Table 6 the elements are displayed which are mandatory according to the IMS metadata specification. The 'minimum level' column states at what level an element can be considered to be required. For example if an element is mandatory at a minimum level of SCO, it will also be mandatory at CAM level, but not at Asset level.

To certify compliance with SCORM, the metadata output needs to be checked for these mandatory elements. A XML file has been built for this reason resembling the XML format specified by SCORM. For every element a 'mandatory' subelement has been provided with an attribute 'level'. The value of 'level' can be 'asset', 'sco' or 'cam'. For every element in the XML file, defining the mandatory elements, the counterpart is searched in the metadata file. If the mandatory element hasn't been found in the metadata file, the check failed. This results in a non conforming SCORM metadata file.

Category number	***Element name***	***Mandatory Type A***	***Mandatory Type B***	***Minimum level if element of Type B***
1.0	general	x		
1.2	title	x		
1.3	catalogentry		x	SCO
1.3.1	catalog		x	SCO
1.3.2	entry		x	SCO
1.5	description	x		
1.6	keyword		x	SCO
2.0	lifecycle		x	SCO
2.1	version		x	SCO
2.2	status		x	SCO
3.0	metametadata	x		
3.4	metadatascheme	x		
4.0	technical	x		
4.1	format	x		
4.3	location	x		
6.0	rights	x		
6.1	cost	x		

Category number	***Element name***	***Mandatory Type A***	***Mandatory Type B***	***Minimum level if element of Type B***
6.2	copyrightandother restrictions	x		
9.0	classification		x	SCO
9.1	purpose		x	SCO
9.3	description		x	SCO
9.4	keyword		x	SCO

Table 6: Mandatory elements as specified by SCORM

4.7.2. Validation of the manifest

ADL provides two XML Schemas for the validation of the manifest file:
1. IMS Schema file,which basically covers the largest part of the manifest
2. ADL Schema file, which validates extensions made by ADL

The IMS Schema file will validate whether the organizational structure, which can act as a menu structure in a LMS, has been properly defined. Also the definitions of the resource used in the aggregation will be validated.
The extensions made by ADL allow the definition of the type of the resources, such as SCO or asset, as well as locations of metadata files.

After loading the manifest in the XML parser from Microsoft, this file is validated.

4.8. Testing the generated SCORM package.

Several tools and applications, which import SCORM packages and are able to run them, are discussed now.

ADL Conformance Suite: ADL has made a conformance suite available on the web, which allows users to test their SCORM package. This will be a great help at testing whether SCORM packages generated in the publication module of Content-e are indeed in accordance with the standard. This Suite is a web application, which will guide the user through some webpages to allow the testing of any aspect of a SCORM package.

ADL Sample Run-time Environment: ADL has made a basic run-time environment of SCORM available. This environment includes a webserver and is again a web application. SCORM packages can be uploaded and viewed in this environment. It is only for testing purposes and not recommended for production use.

Reload SCORM Player: Reusable eLearning Object Authority & Delivery (Reload) has made some tools available to work with content conforming to the SCORM standard. Interesting about the SCORM player is the ability to see, what effect a SCO will have on a LMS. It is possible to see the variables contained in the LMS in a nice overview, but also the change of a variable by a SCO. It will give a better perception than for instance the ADL Sample Run-time Environment. This environment doesn't give an overview of the value of variables in the SCORM Runtime System.

Blackboard, a big player on the learning management systems market, has SCORM support in version 6. It seems to be having the ADL Runtime Environment as implementation for this support.

Hive, a repository, imports SCORM packages, and makes them available in their so-called 'categories'. Hive has an interesting feature: it can extract the metadata from the package, and enable a search on these metadata.

Packages generated by the SCORM publication are valid according to the ADL Conformance Suite. They can also be loaded in Blackboard without any problem.

5. Conclusions

This section presents the conclusion based on the definition and implementation of the mapping process to export content from Content-e to SCORM. The implementation is available in Content-e as the SCORM publication.
The intermediate structure developed in the mapping process can be used in other publication formats of Content-e as well. The Light publication, producing a HTML application, has already incorporated this structure.
Researching the mapping process has lead to suggestions for future improvements. These improvements are discussed below.

Calls to the LMS
Exercises are now built into Content-e, using the XML editor. The score of these exercises can be set in the SCORM runtime system of a LMS. A teacher could then look at an overview of student scores stored in this system.

Resource Management
Resources don't exist as objects in Content-e, which prevents them from having properties attached to them. This prevents the SCORM publication from incorporating metadata for assets.
It also prevents the detection of deletion of files in Content, causing links in the content to point to nowhere instead of the desired files. Resource management would warn the user, if he was trying to delete a file on which paragraphs depend. The paragraphs contain links to this file.

Flexible SCO
Flexible grouping of documents in the SCORM publication allows SCO with multiple pages. A SCO is now set to a single web page. Navigation would have to be provided between these pages.

Supplying values for properties with forms
An implementation has been made to enter properties in forms with the XML editor. This allows more control over the properties and assures property structures are properly created. Storage of the properties with XML editor needs to explored further to allow integration with Content-e.
For quite simple property structures the current property module in Content-e suffices, but larger structures such as used for SCORM pose difficulties.

The last mentioned improvement should be explored first to allow the SCORM publication to be used by a broader user base.

6. References

[Dodds, 2001] P. Dodds. *The SCORM Overview.* 2001.

[Florescu, 1999] D. Florescu, D. Kossmann. *Storing and Querying XML Data using an RDMBS.* IEEE Data Engineering Bulletin, Vol 22, p 27-34. 1999.

[Lee, 2001] D.Lee, M. Mani, F. Chiu, W.W. Chu. *Nesting-based Relational-to-XML Schema Translation.* 2001.

[Lee, 2002] D. Lee, M. Mani, F. Chiu, W.W. Chu. *NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints.* 2002.

[Poel, 2002] B. v.d. Poel. *XML Editor from design to implementation.* 2002.

[Sattler, 2003] K. Sattler, S. Conrad, G. Saake. *Interactive example-driven integration and reconciliation for accessing database federations.* 2003.

[Valkenburg, 2004] W.F. van ValkenBurg, J.G.M. Hamers. *Learning Content Management en de kunst van de ordening.* 2004.

[Shanmugasundaram, 2001] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, J. Funderburk. *Querying XML Views of Relational Data.* 2001.

Appendix A.XSchema and XML Schema of intermediate XML format

A.1.XSchema

Menus(me_introdoc_id) $\subseteq$ Documents(do_id)
MenuItems(me_id) $\subseteq$ Menus(me_id)
M(Menus) = (me_id, me_name, me_description?, me_startmessage?, MenuItems*)
A(Menus) = {Ref_Documents}

MenuItems(mi_reference_id) $\subseteq$ Menus(me_id)
MenuItems(mi_document_id) $\subseteq$ Documents(do_id)
MenuItems(mi_parent_id) $\subseteq$ MenuItems(mi_id)
M(MenuItems) = (mi_id, mi_label, MenuItems*)
A(MenuItems) = {Ref_Documents}, *A*(MenuItems) = {Ref_Menus}

Content(do_id) $\subseteq$ Documents(do_id)
M(Documents) = (do_id, do_document_name, do_document_description?, Content*)

Content(co_parent) $\subseteq$ Content(co_id)
M(Content) = (co_id, co_content, co_parent, co_url?, Content*)

Properties(parent) $\subseteq$ Properties(id)
M(Properties) = (id, name, value?, co_url, Properties*)

'Menus', 'Documents' and 'Content' can contain multiple properties, the properties can however not be shared between those entities, this is defined by the logic of Content-e. Properties should therefore be made into subelements instead of IDREFs. This changes the schemas for those tables:
M(Menus) = (me_id, me_name, me_description?, me_startmessage?, Properties*, MenuItems*)
M(Documents) = (do_id, do_document_name, do_document_description?, Properties*, Content*)
M(Content) = (co_id, co_content, co_parent, co_url?, Properties*, Content*)

Some unnecessities in names are also going to be removed such as prefixes. 'me_id' becomes 'id', 'do_document_name' will be referenced as 'name'.

A.2.XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- *************************** -->
  <!-- ** Attribute Declaration ** -->
  <!-- *************************** -->
  <xsd:attributeGroup name="attr.id">
  <xsd:attribute type="xsd:positiveInteger" name="id"
use="required"/>
  </xsd:attributeGroup>
  <!-- ******************* -->
  <!-- ** Complex Types ** -->
  <!-- ******************* -->
  <xsd:element name="publication">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="menus">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="menu" type="menuType"
maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
          <xsd:keyref name="introdocumentRef" refer="documentKey">
            <xsd:selector xpath="menu"/>
            <xsd:field xpath="introdoc_reference"/>
          </xsd:keyref>
        </xsd:element>
        <xsd:element name="documents">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="menu" type="menuType"
maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <!-- all menus should be unique -->
    <xsd:unique name="uniqueMenu">
      <xsd:selector xpath="menus/menu"/>
      <xsd:field xpath="@id"/>
    </xsd:unique>
    <!-- all documents should be unique -->
    <xsd:unique name="uniqueDocument">
      <xsd:selector xpath="documents/document"/>
      <xsd:field xpath="@id"/>
    </xsd:unique>
    <!-- enable references to documents -->
    <xsd:key name="documentKey">
      <xsd:selector xpath="documents/document"/>
      <xsd:field xpath="@id"/>
    </xsd:key>
    <!-- enable references to menus -->
    <xsd:key name="menuKey">
      <xsd:selector xpath="menus/menu"/>
      <xsd:field xpath="@id"/>
    </xsd:key>
  </xsd:element>

  <!-- create an explicit reference to a existing document -->
  <xsd:complexType name="menuType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="type" type="xsd:integer"/>
      <xsd:element name="description" type="xsd:string"
minOccurs="0"/>
      <xsd:element name="startmessage" type="xsd:string"
minOccurs="0"/>
      <xsd:element name="introdoc_reference" minOccurs="0"
type="referenceType"/>
      <xsd:element ref="properties" minOccurs="0"/>
      <xsd:element ref="menu_item" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="attr.id"/>
  </xsd:complexType>

  <xsd:element name="menuitem">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="label" type="xsd:string"/>
        <xsd:element name="menu_reference" type="referenceType"
minOccurs="0"/>
        <xsd:element name="document_reference" ref="referenceType"
minOccurs="0"/>
        <xsd:element ref="menuitem" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="attr.id"/>
    </xsd:complexType>
    <!-- create an explicit reference to a existing document -->
    <xsd:keyref name="documentRef" refer="documentKey">
      <xsd:selector xpath="."/>
      <xsd:field xpath="document_reference"/>
    </xsd:keyref>
    <!-- create an explicit reference to a existing menu -->
    <xsd:keyref name="menuRef" refer="menuKey">
    <xsd:selector xpath="."/>
      <xsd:field xpath="menu_reference"/>
    </xsd:keyref>
  </xsd:element>
  <xsd:element name="document">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string" minOccurs="0"/>
        <xsd:element name="description" type="xsd:string"
minOccurs="0"/>
        <xsd:element ref="properties" minOccurs="0"/>
  <xsd:element ref="content" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      <xsd:attributeGroup ref="attr.id"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="content">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="style" type="xsd:string" minOccurs="0"/>
        <xsd:element name="list_type" type="xsd:string"
minOccurs="0"/>
        <xsd:element name="embed_children" type="xsd:string"
minOccurs="0"/>
        <xsd:element ref="properties" minOccurs="0"/>
        <xsd:element name="text" type="xsd:string"/>
        <xsd:element ref="content" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="attr.id"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="properties">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="property" type="propertyType"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="propertyType">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="type" type="xsd:string"/>
      <xsd:element name="value" type="xsd:string" minOccurs="0"/>
      <xsd:element name="property" type="propertyType"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="attr.id"/>
  </xsd:complexType>
  <!-- ***************** -->
  <!-- ** Simple Types ** -->
  <!-- ***************** -->
  <xsd:simpleType name="referenceType">
    <xsd:restriction base="xsd:positiveInteger"/>
  </xsd:simpleType>
</xsd:schema>
```

Example 17: XML Schema for intermediate structure

Appendix B.UML diagram XMLWriter

In Figure 23 a UML diagram is shown for the XMLWriter class and the classes already written for Content-e in ASP. The XMLWriter has been designed to create a XML object out of the data stored in the Content-e database. This XML object is the intermediate XML structure provided with data.
Parameters of the functions in this diagram have been left out, due to layout difficulties and are mentioned in the text below.

B.1.Class XMLWriter

The class XMLWriter has the following methods:

- **`setDocument(intDocID, blnAddComments, blnFlatProperties)`**
  Description: set a document as the start for data retrieval
  Parameters:
  `intDocID:` the id of a document.
  `blnAddComments:` whether to include comments, created in Content-e, in the output.
  `blnFlatProperties:` whether to output the properties in a flat list or in a tree-like structure.

- **`setMenu(intMenuID, blnFlatMenu, blnMenuRecurse, blnAddComments, blnFlatProperties)`**
  Description: set a menu as the start for data retrieval
  Parameters:
  `intMenuID:` the id of a menu.
  `blnFlatMenu:` whether to output recurring menu objects in a flat list or in a tree-like structure.
  `blnFlatProperties:` whether to output the properties in a flat list or in a tree-like structure.

- **`loadStructure()`**
  Description: load menu, document, paragraph (ContentRecord in the figure) and properties into a XML object

- **`getXMLObject()`**
  Description: return an XML object.

- **`getXMLString()`**
  Description: return a XML string.

The following classes in this appendix already existed in Content-e.

B.2.Class Menu

The class Menu has the following methods:

- **`Menu(objParent, intMeID, blnRecurse)`**
  Description: constructor

Parameters:
`objParent`: the parent object of this Menu.
`intMeID`: Id of the menu.
`blnRecurse`: whether to traverse through child Menu objects.

- **`getDocuments()`**
  Description: return an array of document ids of the documents referenced by this Menu.

- **`getItems()`**
  Description: return an array of ids of MenuItem objects used in this Menu.

- **`getMenuItems()`**
  Description: return an array of MenuItem objects used in this Menu.

- **`getProperties()`**
  Description: return an array of Property objects used in this Menu.

B.3.Class MenuItem

The class MenuItem has the following methods:

- **`MenuItem(objParent, intMiId, intParentId, intChild, strMiLabel, intReferenceId, intDocumentId, intLevel, blnRecurse)`**
  Description: constructor
  Parameters:
  `objParent`: the parent object of this MenuItem object.
  `intMeId`: Id of the Menuitem.
  `intParentId`: Id of the parent MenuItem of this Menuitem.
  `intChild`: number of child MenuItem objects.
  `strMiLabel`: label of the MenuItem.
  `intReferenceId`: id of the Menu object referenced by this MenuItem object.
  `intDocumentId`: id of the Document object referenced by this MenuItem object.
  `intLevel`: level in hierarchy of menu structure.
  `blnRecurse`: whether to traverse through child Menu objects.

B.4.Class Document

The class Document has the following methods:

- **`Document(objParent,intDoId, strLabel)`**
  Description: constructor
  `objParent`: the parent object of this Document object.
  `intDoId`: Id of the Document.
  `strLabel`: label of the Document.

- **`getData(strRelMM, intCoId, blnGetComments)`**
  Description: load contents of the Document.
  `strRelMM`: a relative path, used for specifying references in paragraphs of Document.

`intCoId`: Id of the top paragraph, from which all child paragraphs are retrieved.
`blnGetComments`: whether to load comments added to paragraphs.

- **`getProperties()`**
  Description: return an array of Property objects used in this Document.

B.5.Class ContentRecord

The class ContentRecord, representation of a paragraph, has the following methods:

- **`ContentRecord(rs, strRelMM)`**
  Description: constructor
  `rs`: result set of a database query to retrieve values of the paragraph.
  `strRelMM`: a relative path, used for specifying references in the paragraph
  .
- **`getProperties()`**
  Description: return an array of Property objects used in this ContentRecord.

B.6.Class Property

The class Property has the following method:

- **`Property(propRS)`**
  Description: constructor
  `propRS`: result set of a database query to retrieve values of the property.

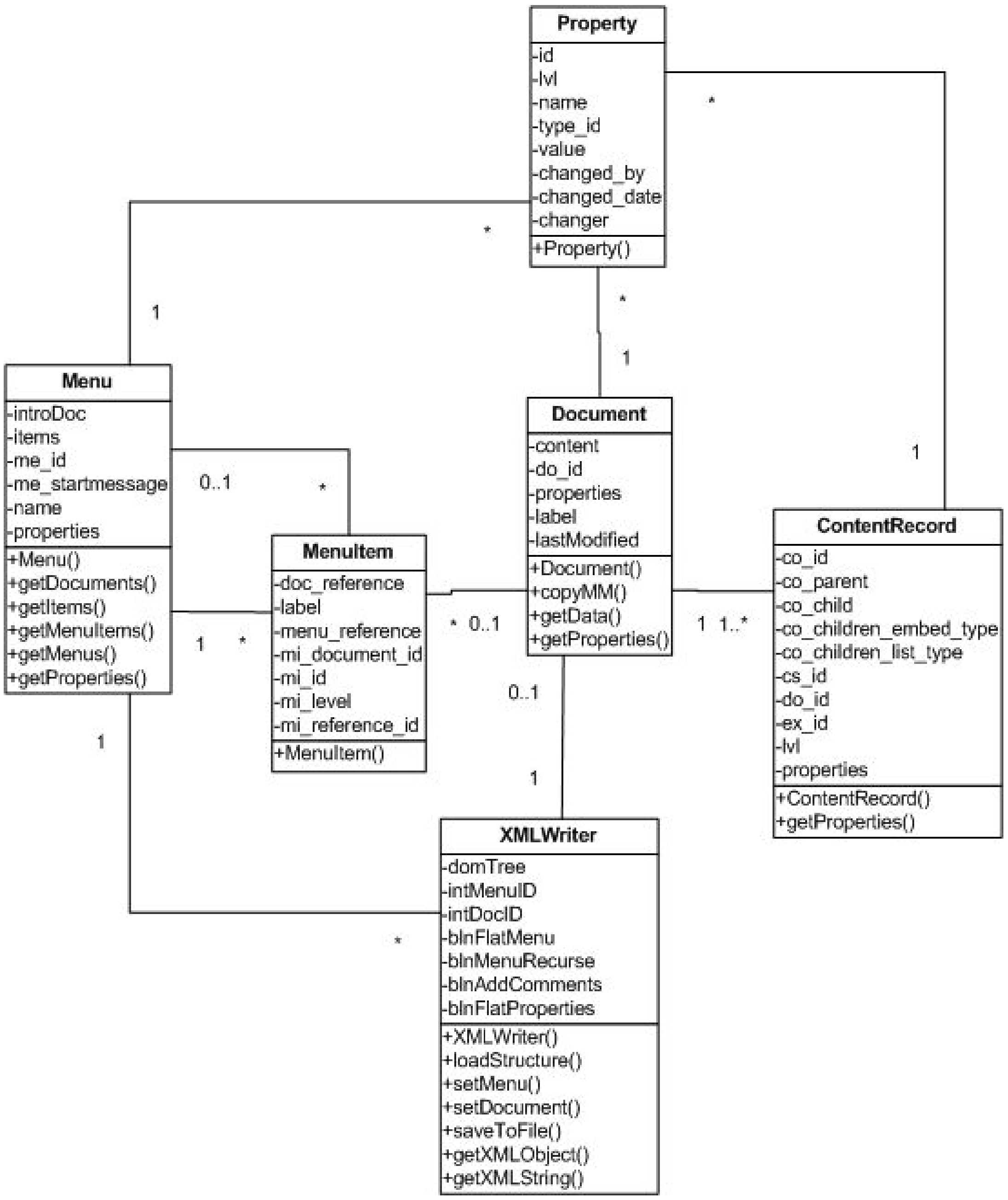


Figure 23: UML diagram XMLWriter and related classes

Appendix C.UML diagram SCORMWriter

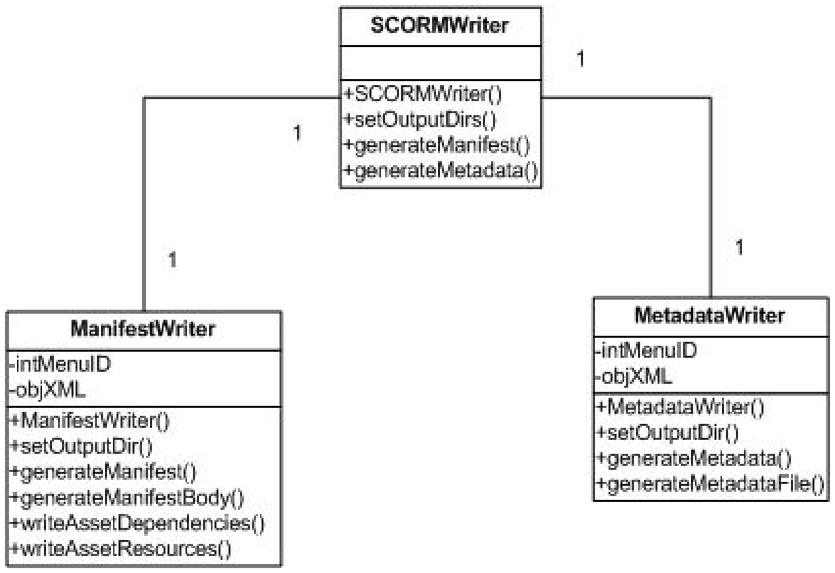


Figure 24: UML diagram SCORMWriter and related classes

C.1.Class SCORMWriter

A class SCORMWriter has been implemented to generate the manifest and metadata files. An UML diagram is shown in Figure 24.

The class SCORMWriter has the following methods:

- **`SCORMWriter()`**
  Description: constructor of the class, initialize internal variables.

- **`setOutputDirs (strMetadataBaseDir,strMetadataRelativeDir,strManifestDir)`**
  Description: set output paths of generated files
  Parameters:
  `strMetadataBaseDir`: the path to the id of a document.
  `strMetadataRelativeDir`: relative path to store the metadata files.
  `strManifestDir`: the path to store the manifest in.

- **`generateManifest()`**
  Description: generate a manifest file.

- **`generateMetadata()`**
  Description: generate all metadata files.

C.2.Class MetadataWriter

The class MetadataWriter has the following methods:

- **`MetadataWriter(objXML)`**

Description: constructor of the class, initialize internal variables.
Parameters:
`objXML`: the XML object representing the XML intermediate structure.

- **`generateMetadata()`**
  Description: generate metadata files for every menu, document in the XML intermediate structure.

- **`generateMetadataFile(intSCORMType, intID, strFileName)`**
  Description: generate a metadata file.
  Parameters:
  `intSCORMType`: the type of object for which the metadata is generated.
  `intID`: the id of the object.
  `strFileName`: name of the metadata file to be generated.

C.3.Class ManifestWriter

The class ManifestWriter has the following methods:

- **`ManifestWriter(objXML)`**
  Description: constructor of the class, initialize internal variables.
  Parameters:
  `objXML`: the XML object representing the XML intermediate structure.

- **`generateManifest()`**
  Description: generate manifest file.

- **`generateManifestBody()`**
  Description: generate a the organization section of the manifest and return a XML representation this section.

- **`writeAssetDependencies()`**
  Description: add dependencies found in the documents in the XML intermediate to SCOs in the manifest.

- **`writeAssetResources()`**
  Description: add assets as resources to the manifest.