5. System Design, Implementation and Testing

This chapter provides a comprehensive illustration of proposed web authoring tool’s technical design, implementation and testing.

5.1. Approach

The main objective of this research is to implement a web authoring tool for generating course materials with the sub research problems identified in the section 1.1 (Objectives and Motivation) of this document. Creating a comprehensive web authoring tool is a complicated and time consuming software project. Thus the emphasis is placed on developing a minimalist web authoring tool which would provide the basic functionality and at the same time addressing the defined objectives.

During the design phase of the application, there was couple of decisions made on the technologies and the tools to be used for the implementation. MS Word 2007 was selected as the editing tool and that decision was due to the following reasons;

- MS Word 2007 has a much superior, user friendly math typesetting tool (Equation Builder) compared to the MS Word 2003 typesetting tool (Equation Editor). MS Word 2007 math equations have high quality display and typography which uses TeX standards making it ideal for creating scientific documents.
- MS Word 2007 is the most recent version of the MS Word; it is becoming popular with the computer user community as a user friendly word processing tool.
- MS Word 2007 uses Open XML publicly documented file format compared to the proprietary MS Word 2003 binary file format, allowing the developers to interoperate much easier. This interoperability avoids the use of Macro programming techniques to extract the content from documents as
most of the existing commercial web authoring tools do with MS Word 2003.

The tool is developed using the Microsoft .NET 3.0 framework. It is the new managed code programming model for Windows which combines the power of the .NET Framework version 2.0 with new technologies for building applications like Presentation Foundation, Windows Communication Foundation, and Windows Workflow Foundation. The .NET Framework 3.0 is included as part of the Windows Vista operating system. But the redistributable package for .NET 3.0 can be used for installing it on Windows XP and Windows server 2003.

One of the objectives of this tool is to create the content offline allowing the users to create course content without connecting to the Internet. Hence this tool is developed as a Windows desktop application. The basic process flow according to above explained approach is displayed in Figure 5.1 below.

As the first step, authors will prepare the course content in MS Word documents according to the guidelines given in Appendix C. If they don’t have the tool to generate the web content, they can send them either electronically (e.g. via email/file upload) or offline (e.g. using magnetic or optical storage devices) to a central location to generate the course content. Then the documents will be transformed as web content using the tool and published in a LMS ready format. These published materials can be reviewed and the feedback can be collected. This collected feedback will be used for next revision of the course content.
5.2. Scope

This section outlines the scope of the current research, highlighting the specific points that are in and out of scope.

5.2.1. Scope Inclusion

The high-level scope of the CourseEditor tool includes a system that provides the ability to do the following:

Course Content: Tool creates the static course content directly from the MS Word 2007 documents. The generated web site includes an index page and a TOC page with the author specific metadata.

Support for Math Equations: Mathematical equations in MS Word 2007 documents are converted to MathML and embedded in the generated web pages as MathML 2.0 islands.

Browser Support: Generated web pages with MathML islands can be viewed in all MathML viewable browsers.

SCORM Compliance: Generated course content is created as an SCO unit according to the SCORM 1.2 specification. However since most of the SCORM 1.3 LMSs expected to support SCORM 1.2, the generated content can be directly use in SCORM 1.3 LMSs.
Enabling Local Language Support: The design of the tool leverage the internationalization features in Microsoft .NET framework. Resource files are used in internationalizing the Windows forms.

5.2.2. Scope Exclusion

The following items are considered out of scope and they will be addressed in the future:

Interactive Course Content: Interactive course content such as self test questions, streaming audio and video, Flash cards will not be able to generate from this tool.

Local Language Support: As per the initial proposal, the scope of the tool contained the local language support. But due to the time constraints it was not implemented in the final product. Hence user input data will not be converted to the local language.

5.3. Design

Simplicity and the extendibility are the main concerns during the design phase of this application. This allows to easily plug-in the additional functionality to the existing tool.

5.3.1. Class Diagram

Figure 5.2 and 5.3 below depict the high level class diagram for the web authoring tool. Diagram in Figure 5-2 has two subsystems for XSL components and Templates. XSL components include all the XSL files used in the tool conversion from one XML format to another. Template subsystem includes all the template files used for generating the web site. For example it contains the TOC and index web pages. This will increase the performance and configurability of the tool since it does not require implementing these common pages from scratch.

WorkBench: Code behind class of the main user interface (Appendix-A Figure A-1) of the application.
**MetaDataPopUp**: Code behind class for the pop up window for entering metadata for course content.

**DocumentFileInfoPanel**: Code behind class for the custom user control which holds the file information of the source document.

**DocumentConstants**: Holds all the application specific constants.

**DocumentEventArgs**: Extended from System.EventArgs class for containing event specific data.

**Document**: Model class containing the Event handlers.

**DocumentEventHandler**: Event handler delegate for the document methods.

**DocumentProcessor**: Main class containing the document processing functionality. This will extract the data in the MS Word document and convert to XHTML using the XSL transformations.

**MathMLParser**: Converts the mathematical equations from the OMML format to MathML using the XSL transformations.

**PageData**: Data object for page specific data

**DocumentMetaData**: Data object for document metadata.
Figure 5-2 - Class Diagram
Figure 5-3 - Class Details
(Original in color)
5.3.2. Sequence Diagram

The following UML diagram in Figure 5-4 illustrates the complete process of generating a web site through the CourseEditor tool.

![Sequence Diagram to Generating a Web Site](image)

As the first step, user specifies the source document through the File menu of the WorkBench (Appendix-A Figure 1) user interface. Next the user optionally specifies the metadata (Appendix-A Figure 2) for the course content to be generated. If specified these metadata will be included in all of the generated web pages. User input on generating SCORM specific content are also optionally handled at this point. Finally the user gives the command to create the content through the Course menu specifying the destination location. On successfully creating the web content the application will acknowledge the user with a message.
5.4. Implementation

Microsoft Visual Studio 2005 is the IDE for developing the CourseEditor tool. This application runs on the Microsoft .NET 3.0 framework; thus it requires the common language runtime which comes with the Microsoft .NET Framework 3.0 Redistributable Package [10].

5.4.1. Technology Stack

In developing this tool, multiple technologies were used. The complete technology stack used during the development phase is listed in the following Table 5-1.

<table>
<thead>
<tr>
<th>Technology</th>
<th>Version</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Studio</td>
<td>2005</td>
<td>Microsoft</td>
</tr>
<tr>
<td>C#</td>
<td>2.0</td>
<td>Microsoft</td>
</tr>
<tr>
<td>.NET framework</td>
<td>3.0</td>
<td>Microsoft</td>
</tr>
<tr>
<td>Windows XP professional</td>
<td>2003</td>
<td>Microsoft</td>
</tr>
<tr>
<td>Office Word</td>
<td>2007</td>
<td>Microsoft</td>
</tr>
<tr>
<td>OMML</td>
<td>1.0</td>
<td>Microsoft</td>
</tr>
<tr>
<td>MathML</td>
<td>2.0</td>
<td>W3C</td>
</tr>
<tr>
<td>XML</td>
<td>1.0</td>
<td>W3C</td>
</tr>
<tr>
<td>XSLT</td>
<td>2.0</td>
<td>W3C</td>
</tr>
<tr>
<td>XPath</td>
<td>2.0</td>
<td>W3C</td>
</tr>
<tr>
<td>CSS</td>
<td>2.1</td>
<td>W3C</td>
</tr>
<tr>
<td>SCORM</td>
<td>1.2</td>
<td>ADL</td>
</tr>
<tr>
<td>JavaScript</td>
<td>1.7</td>
<td>Sun Microsystems / ECMA standard</td>
</tr>
<tr>
<td>OpenXML</td>
<td>1.0</td>
<td>Microsoft / ECMA standard</td>
</tr>
</tbody>
</table>

Table 5-1 - Technology Stack
5.4.2. Packaging Structure

Following Figure 5-5 depicts the folder structure for the project. All source code would be segregated into separate logical folders within the package. These solution folders allow organizing related projects into groups and then performing actions on those groups of projects.

Figure 5-5 - Packaging Structure
(Original in color)
5.4.3. Extracting Data from Source Documents

The Windows Presentation Foundation (WPF) is the graphical subsystem feature of the .NET framework 3.0. It is pre-installed in Windows Vista, the latest version of the Microsoft Windows operating system. WPF is also available for installation on Windows XP SP2 and Windows Server 2003. It provides a consistent programming model for building applications and provides a clear separation between the UI and the business logic.

WPF natively supports reading and writing paginated documents using Open Packaging Convention (OPC), which is a file packaging format created by Microsoft for storing a combination of XML and non-XML files that together form a single entity. OPC is specified in Part 2 of the ECMA Office Open XML standard [7]. OPC files can be opened using common ZIP utilities.

This tool needs to interact with the Open XML file parts while extracting the data from the MS Word 2007 documents. The System.IO.Packaging namespace makes it possible to work with these compressed Open XML file formats. The System.IO.Packaging API resides on the WindowsBase.dll. When a WPF project is created in Visual Studio, it automatically makes references to this WindowsBase.dll. Abstract class Package in System.IO.Packaging namespace is used to hold the zip file of the MS Word document in memory while doing the data manipulation tasks.

5.4.4. Custom Event Handlers

During the implementation, custom event handlers were used to segregate the business logic from the presentation logic of the tool. An event is a message sent by an object to notify other objects that an action has occurred. A method that handles an event is called an event handler. In .NET, any object can publish a set of events to which other classes can subscribe. When the publishing class raises an event, all the subscribed classes are notified. This design is a form of the Observer Pattern
There are two key objects in this pattern: the subject and the observer. The subject may have one or many observers. These observers listen for notification from the subject of a state change inside the subject. This pattern is also known as *publish-subscribe*.

One of the key aspects of .NET programming is the use of delegates to handle events. A delegate is a class that can hold a reference to a method. Unlike other classes, a delegate class has a signature, and it can hold references only to methods that match its signature. A delegate is thus equivalent to a type-safe function pointer, or a callback. The advantage of the publish/subscribe idiom is that any number of classes can be notified when an event is raised. The publisher and the subscribers are decoupled by the delegate. This is highly desirable since it makes for more flexible, robust and extensible code.

### 5.4.5. XSL Transformations

XSL is a family of recommendations for defining XML document transformation and presentation. An XSLT style sheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses a formatting vocabulary, such as XHTML. A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree.

In CourseEditor OpenXML2XHTML.xsl extracts and converts the data from Open XML document to XHTML format. The existing implementation does not support all the MS Word document formats. It supports only the following features of MS Word:

- Mathematical equations (inline and equation paragraphs)
- Titles and headings with alignments
- Paragraphs with alignments
• Tables
• Bullets and numbering

OMML2MML.xsl converts the math equations in OMML into MathML format. The XslCompiledTransform class is the XSLT processor that comes with .NET framework. All the above mentioned transformations use this XSLT processor. The XSLT namespace has the URI http://www.w3.org/1999/XSL/Transform. As listed in Table 5-2 there are multiple XML namespaces involved in the document data extractions process.

<table>
<thead>
<tr>
<th>XML Namespace</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlns:w</td>
<td><a href="http://schemas.openxmlformats.org/wordprocessingml/2006/main">http://schemas.openxmlformats.org/wordprocessingml/2006/main</a></td>
</tr>
<tr>
<td>xmlns:m</td>
<td><a href="http://schemas.openxmlformats.org/officeDocument/2006/math">http://schemas.openxmlformats.org/officeDocument/2006/math</a></td>
</tr>
<tr>
<td>xmlns:mml</td>
<td><a href="http://www.w3.org/1998/Math/MathML">http://www.w3.org/1998/Math/MathML</a></td>
</tr>
<tr>
<td>xmlns:xsl</td>
<td><a href="http://www.w3.org/1999/XSL/Transform">http://www.w3.org/1999/XSL/Transform</a></td>
</tr>
<tr>
<td>xmlns:</td>
<td><a href="http://www.w3.org/1999/xhtml">http://www.w3.org/1999/xhtml</a></td>
</tr>
</tbody>
</table>

Table 5-2 - XML Namespaces in CourseEditor Tool

XmlNodeManager class comes with .NET framework, resolves, adds and removes namespaces to a collection and provides scope management for these namespaces. This stores the prefixes and namespaces as strings in a NameTable class.

XPath is an expression language used by XSLT to access or refer to parts of an XML document. It also provides basic facilities for manipulation of strings, numbers and booleans. In addition to its use for addressing, XPath is also designed so that it has a natural subset that can be used for matching (testing whether or not a node matches a pattern).
XPath models an XML document as a tree of nodes. There are different types of nodes, including element nodes, attribute nodes and text nodes. XPath defines a way to compute a string-value for each type of node. XPath fully supports XML Namespaces (XML Names). Thus, the name of a node is modeled as a pair consisting of a local part and a possibly null namespace URI.

5.4.6. Adding Custom Styles

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts, colors, and spacing) to web documents. CourseEditor generated web sites use CSS to customize the look and feel giving the authors more flexibility and control. Also it enables the separation of document content (written in HTML/XHTML) from document presentation. This separation can improve content accessibility, provide more flexibility and control in presentational characteristics. Also this reduces the complexity and repetition in the structural content.

External style sheets, i.e. a separate CSS-file referenced from the document are used in CourseEditor. Browser specific CSS tags are avoided as far as possible to have cross browser support.

5.4.7. Cross Browser Compatibility

Most of the existing software products render MathML in a browser specific way. This becomes a large obstacle in passing information in MathML for a larger audience. W3C describes a way to include MathML in web pages in a way that makes it possible to be viewed on a large number of browsers [12]. In order to maximize the number of platforms it will be viewable on, there are a set of rules that the web pages should adhere to. First, the web pages should create using XHTML in lined MathML as shown in following Figure 5-6.
The MathML markup should not be in an `<object>` tag, or in a separate file linked from an `<embed>` tag as most of the current plugging software do. Next, each web page should have a style sheet for processing instruction at the beginning of XHTML page as shown in the following Figure 5-7.

This will work in most cases except in IE. For security reasons IE will not execute an XSLT style sheet that is not located on the same server as the XHTML+MathML document. Therefore it requires to explicitly changing the security options of the browser to these XHTML pages to run XSLT style sheets.

Another consideration in viewing MathML in browsers is the available fonts. The MathPlayer plug-in for IE will download and install fonts required for viewing mathematical content. But for Mozilla, users have to install additional fonts since some of the symbols get distorted while viewing through Mozilla FireFox browser. These additional fonts can be downloaded at MIT MathML homepage [13].
5.4.8. SCORM Communication Session

The only two events specified in SCORM to manage the communication session between the SCO and the LMS run time environment are `LMSInitialize` and `LMSFinish`. Calling `LMSFinish` means that the SCO no longer needs to communicate with the runtime service. A SCO can get unloaded at any time when the user chooses another SCO. Also closing the stage window in which a SCO is playing at any time causes an unexpected unloading of SCO. To minimize the harmful consequences of such unexpected unloading, an "onunload" event handler should be included in the top level of the SCO that calls `LMSFinish`.

If unloaded unexpectedly, SCO object attempts to call `LMSSetValue` to communicate any unsaved data to the LMS, and then calls `LMSFinish`. This cleanup should be included in a handler for the `onUnload` event triggered by the unloading of the SCO. Because the behavior of browsers during execution of an `onUnload` can be unreliable, there is no guarantee that the data will effectively be received and stored by the LMS.

A large part of the script in a SCO is devoted to finding the API adapter provided by the LMS in the DOM environment before the SCO can communicate with it. The `(window.parent !== window)` test is necessary to work around an IE oddity.

5.5. Testing

MathML Tester by Mozilla [14] is used for verifying the generated MathML content in CourseEditor. The generated web pages properly render both the inline mathematical contents and the mathematical paragraphs properly (See Appendix B Figure B-1).

The set of integrated e-learning specifications of ADL's Sharable Content Reference Model (SCORM) comes with a do-it-yourself test suite [15]. It is a free
an open source (FOSS) tool which helps consistent adoption by being the one reference implementation everyone tests against. This zip file contains the executable installation program for the SCORM Version 1.2 Conformance Test Suite Version 1.2.7 (Self Test). SCOs created by the CourseEditor tool are tested through this self test tool.

The test suite provides a common framework for all of the conformance tests. Each test includes a step-by-step set of detailed instructions as well as a log of each test action and the outcome. The test suite runs entirely within a web browser on one local computer. All of the software is loaded on the local hard-drive and requires no HTTP web server to run.

5.5.1. LMS Run-Time Environment (LMS RTE) Conformance Test

The LMS RTE Conformance Test verifies that a test subject LMS is conformant with The Run-Time Environment section of the SCORM. This test will verify that the test subject is able to:

- Launch known conformant SCOs
- Support the Application Program Interface (API) functions defined in the SCORM
  - LMSInitialize"
  - LMSFinish"
  - LMSCommit"
  - LMSGetValue"datamodel.group.element"
  - LMSSetValue"datamodel.group.element","val"
  - LMSGGetLastError
  - LMSGGetErrorString
  - LMSGGetDiagnostic
- Determine if all mandatory data model elements are implemented correctly by the LMS
• Determine which optional data model elements are supported by the LMS and implemented correctly.

5.5.2. SCO Run-Time Environment (SCO RTE) Conformance Test

The SCO RTE Conformance Test verifies that the test subject SCO is conformant with the Run-Time Environment section of the SCORM. This test will verify that the test subject:

• Can be launched by a known conformant LMS

• Supports the API functions defined in the SCORM

  Required
  
  o LMSInitialize(""")
  o LMSFinish("""")

  Optional
  
  o LMSCommit(""")
  o LMSGetValue("datamodel.group.element")
  o LMSSetValue("datamodel.group.element","val")
  o LMSGetLastError()
  o LMSGetErrorString()
  o LMSGetDiagnostics()

In addition to verification of the above requirements, this test will also audit which CMI data model elements the SCO(s) made use of in the content implementations, and verify that the data elements are used in the correct manner with regard to the data type rules associated with the data model elements.