INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600
NOTE TO USERS

This reproduction is the best copy available.

UMI
Connexions: An Architecture for Web-based Educational Materials

by

Brent M. Hendricks

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE Master of Science

APPROVED. THESIS COMMITTEE:

[Signatures]

Richard G. Baraniuk. Chair
Professor of Electrical and Computer Engineering

Don H. Johnson.
Abercrombie Professor of Electrical and Computer Engineering

William L. Wilson.
Professor of Electrical and Computer Engineering

Houston, Texas
January, 2001
Connexions: An Architecture for Web-based Educational Materials

Brent M. Hendricks

Abstract

This thesis describes Connexions, a powerful new architecture for managing web-based educational materials. Connexions, consisting of five components, enables course instructors to take advantage of the cross-linked nature of the Web and helps students visualize the relationships between concepts. The first component is a pool of interlinked content modules written in the Connexions Markup Language (CNXML), an XML-derived language for educational content. Each module covers a single concept and links to related concepts. The second component is a central repository to store and provide access to the modules. The third component consists of tools to assist module authors in creating content and interfacing to the repository. The fourth component is a course construction tool that allows instructors to author course materials by selecting and customizing a sequence of content modules. The fifth component is a navigational tool that shows students their location along the course-path.
Acknowledgments

The funding for this research was provided by Texas Instruments and the National Science Foundation. I would like to thank the members of my thesis committee: Dr. Richard Baraniuk, Dr. Don Johnson, and Dr. William Wilson for their vision in launching Connexions, for giving me the opportunity to work on such an enjoyable and rewarding project, and for their advice and suggestions in preparing this thesis. Many thanks also go to the project's Executive Director Joanne Sonin and to all of the Connexions project students whose hard work and countless hours spent in design meetings helped to make this thesis possible. I would also like to express my gratitude to my parents Floyd and Deborah, who support me in everything I do. I especially want to thank my wonderful wife Megan: without her loving sacrifice, patience, and encouragement I would never have been able to complete this task. Finally, I would like to thank the Lord God, who has guided me and blessed me throughout this thesis and throughout my life.
Contents

Abstract ii
Acknowledgments iii
List of Illustrations vi

1 Introduction and Background 1

2 Requirements 4

3 Architecture 7
   3.1 The Architecture ........................................ 7
   3.2 System Views ........................................ 9

4 Component M: Content Modules 11
   4.1 Source Format: XML .................................... 11
   4.2 Namespaces ........................................ 15
   4.3 Marking up Mathematics ................................ 15
   4.4 Displaying CNXML using Style Sheets ............... 17

5 Component R: Module Repository 19
   5.1 Module Storage ....................................... 19
      5.1.1 Where to store the modules ...................... 19
      5.1.2 How to store the modules ...................... 21
   5.2 Module Access ....................................... 22


## Illustrations

3.1 Four components of the Connexions architecture ............................................. 8
3.2 Module author’s view of the Connexions system .............................................. 9
3.3 Course instructor’s view of the Connexions system .......................................... 10
3.4 Student’s view of the Connexions system ....................................................... 10

4.1 Using multiple style sheets with a single document ....................................... 13
4.2 Example module written in CNXML ............................................................. 14
4.3 Two different presentations of a CNXML module ............................................ 18

5.1 Distributed proxy cache system ...................................................................... 20
5.2 Server side module processing ....................................................................... 22
5.3 Client side module processing ....................................................................... 23

7.1 Module pool with and without course-path .................................................... 29

8.1 Prototype implementation of Roadmap course and chapter views ................. 33
8.2 Prototype implementation of Roadmap module view ..................................... 34
Chapter 1

Introduction and Background

Over the past five years, the World Wide Web has revolutionized the way people store and retrieve information. Websites can be cross-linked and interactive, creating content that is non-linear and dynamic. However, most educational materials online today are merely digitized versions of their linear paper counterparts. This unnaturally forces the limitations of paper onto a medium where no such restrictions exist. This thesis describes the Connexions system, a powerful new architecture for managing educational materials in a way that maximizes the Web's unique advantages. Connexions uses cutting-edge technologies to allow instructors to easily author and customize course content, and to help students visualize the relationships between educational concepts.

The earliest notions of hyperlinked text date back to the 1960s when Ted Nelson founded the Xanadu project [1] to rethink electronic document storage and display. His main concern was how to represent links: both between documents that contained identical materials and between content that was merely related. The Xanadu project envisioned a world where documents would be addressed by their content and not by their physical location, where easily reusable text would be updated automatically from its source, and where documents would refer to each other using bidirectional links. A full implementation was slow in coming, however. The World Wide Web, partially inspired by Nelson's ideas, did not appear until the early 1990s, and still fell short of the Xanadu vision [2].

Although the Xanadu project continued to work towards its goals and release some proof-of-concept software, the Web rapidly gained in popularity and continues
to grow. It has been estimated that almost eight million sites [3] comprise the Web and that over two billion individual web pages currently exist [4]. The technology caught on suddenly and has taken many companies and organizations by surprise. Even universities and research institutions where the Web first took hold seem unsure of how to take advantage of this new tool. In particular, the question of how best to use the Web as a teaching tool remains unanswered.

In 1995, researchers at the Texas A&M Center for the Study of Digital Libraries (CSDL) began work on the Walden’s Paths project [5]. The goal of this project was to provide tools to "enable and facilitate tailoring" of existing web materials for use in a course. Their software allowed instructors to specify a series of websites to be visited in sequence, along with some text to accompany with each site. This yielded an annotated path through the Web, which students could navigate using a provided Java applet. If they decided to leave the path to follow links on any of the sites, the applet would allow them to return easily to the path again. While Walden’s Paths provided a significant advancement in the use of web materials for teaching courses, it still had shortcomings. Instructors could only define linear paths without branches and could only annotate whole webpages, rather than a single line or paragraph. More fundamentally though, there was no way to indicate relationships between materials other than sequentially.

The Open Journal project [6], also launched in 1995, approached this problem by using a Distributed Link Service to store links in "linkbases" separate from the main document. This technology, developed by Southampton University [7], allowed users to add links between documents even if they did not own the materials. For the Open Journal project, this meant that links could be added and maintained without modifying the original document. This helped facilitate the management of relationships between documents and was of great importance to Open Journal. The project recognized that linking was a key advantage that online publications held over traditional printed forms. This also gave them an advantage over Walden’s
Paths because documents in the Open Journal could specify arbitrary relationships and not just sequential ones. Unfortunately, Open Journal researchers neglected to provide a navigational tool like Walden's Paths' Java applet. They did acknowledge this mistake, however, as their user testing demonstrated that people tended to get lost in the sea of links without a clear indicator of their location [8].

Development in the electronic publishing of educational materials has not been confined to the academic world. In recent years several companies have come out with products that attempt to address some of these needs. Companies like Ebooks.com sell textbooks and materials in electronic format for use with hand-held readers. These books, however, are not targeted at online use. They contain no links and are little more than traditional texts in a new cover. Blackboard Inc. provides a complete course management suite with tools to manage course communication, assignments, and even grading. It has facilities for distributing materials online, but the materials are expected to be standalone documents for download, not online materials. More promising is the work being done by relative newcomer Gemini Inc. Their SWIFT course development software helps instructors create online materials which can be viewed by students using their custom navigational tool. While this helps students keep track of their position in the course, there is no facility for showing relationships between materials.

Each of these projects attempts to address one or more of the problems of publishing and using educational materials online. Some possess useful navigational tools and others emphasize link relationships. None of them combines all of the features necessary for truly useful web-based educational materials.
Chapter 2

Requirements

The goal of the Connexions project is to create a powerful web-based system for publishing and accessing educational content and to address the shortcomings of other online educational efforts. The following are desired features of the Connexions system:

R1. A modular, non-linear approach to content

Many electronic publishing systems merely digitize existing texts and do not take advantage of the inherent cross-linked, nonlinear nature of the World Wide Web. This squanders an enormous potential benefit, as the nonlinear model is much closer to the way people learn and store information: by forging connections between new pieces of information and the knowledge they already possess. To take advantage of this, the Connexions system needs to structure information in a modular fashion. It should present each concept as a self-contained unit or module and illustrate how the concept connects to other, related topics. Taken together, these interlinked modules will form a non-linear pool of information. This scheme will help students visualize relationships in the educational curriculum, both within courses and between courses. This last point is a key desired feature for the Connexions system, since students oftentimes do not grasp how different courses interrelate until well after they have graduated.

R2. Simple content creation

If it is too difficult to create materials for the Connexions system, authors will simply avoid it. Connexions should either provide simple content editing tools
or work with existing software. The project should also address the issue of converting content from other formats.

R3. Facilities for defining course-paths
The non-linear pool of modules is an excellent way to show content relationships, but it may be bewildering to know where to begin. Instructors will want to provide their students with a linearization or sequence to follow. The Connexions system must give them the ability to plot a path for their course through the module pool. While these paths will generally be linear, Connexions should also allow for branching and parallel tracks.

R4. Fine-grained annotations
In addition to defining a sequence of modules for their students to follow, instructors may want to add their own comments or explanations to the existing content. Connexions should enable instructors to annotate materials down to the level of individual words as well as providing transition material between modules. Combining this with a custom path will allow instructors to tailor the Connexions system for their particular courses.

R5. Generation of customized printed versions
Although the Connexions system will be based online, many students and professors will want to have a printed copy of the materials for a course. While a simple solution might be to print out each module in sequence, the quality of browser printer output is often poor and certain aspects may need to be presented differently (e.g. interactive demonstrations). Connexions should provide a way to create high-quality printed materials as well as web-based versions.

R6. Online navigational tools
While a non-linear content scheme provides educational benefits, it raises issues of comprehensibility, since there no longer exists a single path to follow through the information. To address this, Connexions must provide tools to assist in
navigating the sea of modules. Such tools should help students see where they are and how the current module relates to other modules. In particular, the navigational tools should display the path defined by the course instructor and allow students to return easily to the path if they decide to follow links to other modules.
Chapter 3

Architecture

To satisfy the requirements listed in Chapter 2, we propose the following architecture for the Connexions system. The system is divided into five components: the content modules, a central module repository, module authoring tools, the course creation tool, and the module navigator/roadmap tool.

3.1 The Architecture

M: Content modules

Chapter 4 describes this component, which consists of a series of interlinked content modules as described in requirement R1. Each module covers a particular topic or concept, and links to other related modules. Other components do not interact directly with component M as a whole, but they may pass individual modules amongst themselves.

R: Central module repository

Component R, described in Chapter 5, is the repository that houses the modules. Other components needing to access modules do so by communicating with component R (see Figure 3.1).

A: Content authoring

This component consists of tools to aid authors in creating content modules and interfacing with the repository to submit the modules to the Connexions system. Component A enables simple content creation, meeting requirement R2, and is detailed in Chapter 6.
Figure 3.1: Four components of the Connexions architecture (content modules component not shown)

C: Course creation

Chapter 7 explains the course creation component. This component allows instructors to select a sequence of modules and add personal annotations, creating a customized course. This satisfies requirements R3 and R4. Instructors can store the course path and annotations on a local web-server for student access, or create a printed version of the linearized course materials, as specified in requirement R5.

N: Module navigator/roadmap

To navigate through the sea of modules, students will use component N, the roadmap, described in Chapter 8. As mentioned in requirement R6, this tool will display module relationships and enable students to view their location along the course-path.
3.2 System Views

The previous section describes the Connexions system as a whole. Most users of the system, however, do not interact with the entire Connexions architecture, but see different views of the system depending on their current task. Module authors create content and interact with the system using the authoring component (see Figure 3.2). For details on how the content authoring component interacts with the repository, see Section 6.2.

![Diagram showing module author's view of the Connexions system](image)

Figure 3.2: Module author's view of the Connexions system

Instructors creating course materials interact with the Connexions system via the course creation component (see Figure 3.3). They obtain information about the modules (metadata) from the repository, and use it to create customized course content. They can then present this content to students in book format or over the web (see Section 7.2).

Students in a course make use of the navigational component to access the Connexions system (See Figure 3.4). This component accesses course information stored on the local course server, and uses that information to display content modules from the central repository.
Figure 3.3: Course instructor's view of the Connexions system

Figure 3.4: Student's view of the Connexions system
Chapter 4

Component M: Content Modules

This component of the Connexions architecture consists of hyperlinked content modules. The most important issue for this component is the choice of source format (i.e., the format or language in which it stores modules). This decision affects the ease of revising materials and heavily influences the tools that can be used to display and manipulate the content.

4.1 Source Format: XML

We have opted to use the Extensible Markup Language (XML) [9] to store content. XML is a language defined by the World Wide Web Consortium (W3C), the body responsible for HTML and other web technologies. XML is a subset of the Standard Generalized Markup Language (SGML) in common use in the publishing industry. While SGML has many of XML’s desirable properties, it is difficult to implement fully and correctly. The W3C designed XML with simplicity in mind, placing a high priority on performance over the web [10].

XML has several advantages over alternative formats. One of the most significant benefits is that it enables and encourages the separation of presentation and content through its use of semantic markup. Semantic markup conveys information about what the data is or means, rather than how it should be displayed. For example, consider the following text written in HTML, the syntactic markup language in common use on the web today:

I just read <u>Thesis Writing for Dummies</u>.

It was <u>great</u>!
Why did the author use underlining here? We could probably guess that in the first case he was marking the title of a book. and in the second case he was merely emphasizing a word. However, we had to rely on contextual clues to make this assessment, and a computer would be hard-pressed to come to the same conclusion. Look at the same text marked up semantically:


It was <emphasis>great</emphasis>!

Now there is no question of the author's intent. The markup tags help convey the meaning of the text, not simply how to display it.

Since the markup contains little or no presentation information, the author must create a separate file called a *style sheet* to detail how the document should be displayed. This approach has several advantages. Perhaps the most obvious is that if the author changes his mind and decides that book titles should be italicized rather than underlined, he need only make one change in the style sheet, not several changes scattered throughout the document. Another advantage is the ease of searching: rather than resorting to a brute-force text-search, users can search directly for the relevant content. Another benefit is the ability to display the content in multiple formats without maintaining it separately for each format. For example, in Figure 4.1 the same document is formatted for presentation on a desktop computer, a hand-held device, a printed page, and even as spoken text.

While it is a relatively new technology, tools and support for XML are readily available, as it is rapidly becoming an industry standard for exchanging information. Tim Bray, one of the editors of the XML 1.0 specification, has said that "XML is the ASCII of the future." [11]. Just as ASCII allows different computers to communicate using a standard encoding of letters into bits, XML enables myriad applications to exchange data using a common language.

XML also has the advantage of being an open, cross-platform standard. The Connexions project aims to reach the largest possible audience and cannot afford
to tie itself to one particular platform or vendor. By avoiding proprietary formats such as Microsoft Word [12] or Adobe’s Portable Document Format (PDF) [13], we ensure that a wide variety of tools from a number of sources and on several different architectures will be available to module authors. Finally, XML documents are human-readable ASCII text files. While specialized XML editors exist, they are not strictly necessary, and module creation and editing can be done by hand.

XML does not itself define tags for marking up documents, but rather establishes rules to create new languages, each with its own set of tags. We have defined the Connexions Markup Language (CNXML) for use as the primary source language in Connexions modules. Figure 4.2 displays an example Connexions module written in CNXML (see also the complete CNXML specification [14]).

The module tag serves as the top-level tag or root element in CNXML documents, which are also called modules. The next several tags comprise the header portion of the document and present information about the document, such as the authors’ names and key words. This is sometimes called data about data, or metadata. The section tag encloses the primary content of the document, in this case an ordered list of items to purchase at the store. CNXML has several tags for basic document markup, including things like tables, paragraphs and links.
<?xml version="1.0" standalone="no"?>
<!DOCTYPE module SYSTEM "http://cnx.rice.edu/cnxml/0.1/DTD/cnxml.dtd">

<module name="Example Module"
        id="example1"
        levelmask="0"
        dateofcreation="2000-10-29"
        dateoflastrevision="2001-01-01">

  <authorlist>
    <author name="Brent M. Hendricks" id="bmh"/>
  </authorlist>

  <keywordlist>
    <keyword>CNXXML</keyword>
    <keyword>Sample Module</keyword>
    <keyword>Connexions Project</keyword>
  </keywordlist>

  <section id="sec1" name="First Section">
    <para id="par1">
      This is a grocery list:
      <list id="groceries" type="enumerated">
        <item>A loaf of bread</item>
        <item>A container of milk</item>
        <item>A stick of butter</item>
      </list>
    </para>
  </section>
</module>

Figure 4.2: Example module written in CNXML
4.2 Namespaces

We readily acknowledge that CNXML cannot define tags for everything that module authors might want to include. Most academic fields have their own jargon for which tags might be desired, and CNXML cannot possibly include them all. Some disciplines already have markup languages, and we certainly do not want to duplicate their efforts. What is needed then, is a way to embed multiple markup languages in a single document. The essential difficulty in using multiple languages together is that some tags might have meaning in more than one language. For example, the \texttt{<table>} tag has a different meaning in HTML than in a language describing office furniture.

Fortunately, the W3C foresaw this difficulty and put forth the recommendation \textit{Namespaces in XML} [15] (W3C standards are called "recommendations"). An XML namespace consists of all of the tags for a given language along with rules for their use. To use a particular namespace, an XML document must assign it a unique identifier and then add the identifier as a prefix to all tags from the given namespace. For example, a document using tags from the HTML namespace might assign the identifier "htm" to those tags such as \texttt{<htm:table>} or \texttt{<htm:b>}. For convenience, XML documents may declare a default namespace so that any tags without namespace identifiers will be assumed to be from the default.

By utilizing namespaces, we can focus on creating a markup language containing only the necessary tags for marking up educational content. Content authors in various academic fields may embed tags from other XML languages as they see fit.

4.3 Marking up Mathematics

Many disciplines (including most branches of science and engineering) make heavy use of mathematics, and so the ability to display equations and mathematical symbols is crucial to Connexions. While \LaTeX{} [16] and PostScript [17] provide attractive printed formats, modern web browsers are incapable of displaying them on-screen. Many instructors work around this problem by creating images of the equations and
embedding the image files in their web pages. This "solution," however, brings with it many problems. First, such images are limited to a specific resolution and font-size, and so the equations do not scale as users alter the font-size in their browsers. Second, the process of correcting errors and making updates becomes significantly more involved. Finally, on a more subtle level the document has traded a meaningful equation for a visual representation, and the equation cannot easily be recovered.

Connexions solves these problems by using a relatively new standard for representing mathematical content: the Mathematical Markup Language (MathML) [18]. MathML is the W3C Recommendation for using XML to store and display math. especially on the Web. Since MathML is an XML-based language, it fits seamlessly into the Connexions system. The language is divided into two sets of tags: Content MathML tags and Presentation MathML tags. Presentation MathML tags detail how to display mathematical notation, while Content MathML tags focus on the meaning of the mathematics that the notation represents.

Since the Connexions project is concerned with semantic markup, we have chosen to focus our efforts on Content MathML. The argument here parallels that for semantic markup in general: it allows us to capture the meaning of an equation, rather than just its notation. The notation can be defined in a style sheet and altered separately from the document. This capability is crucial to the Connexions project since instructors who define course-paths by selecting modules from different authors will still want to maintain notational consistency throughout the course. Even the simple concept of a vector can be represented several different ways. For example: \( \mathbf{x} \), \( \vec{x} \), and \( \dot{x} \) might all be used by different authors to mean the same thing. In Presentation MathML these three different notations would be expressed in three different ways, but using Content MathML we can describe the vector once and use a style sheet to determine how it will be presented. The course composer tool (see Section 7.1) allows instructors to define a style sheet for their course, enabling them to use their preferred notation without rewriting the modules.
Presentation MathML requires the reader to infer the meaning based on notation, whereas Content MathML explicitly states the mathematical meaning. This gives MathML an advantage over even \LaTeX, which tends to be more presentation oriented. For example, a module with embedded MathML could communicate with Wolfram Research’s Mathematica [19] software and allow students to manipulate the equations. This provides a tremendous educational benefit as students can perform hands-on experiments with the formulas.

Although neither of the two leading web-browsers (Internet Explorer [20] and Netscape [21]) directly supports MathML at present, the work that has been done is encouraging. Internet Explorer can render MathML via the techexplorer [22] plug-in from IBM, although it requires the use of non-standard tags to do so. Researchers at IBM are working to remove this restriction. More promising, however, is the MathML work being done for Mozilla [23], the open-source browser that forms the basis for Netscape 6. Available in source format, Mozilla can be compiled to support MathML directly. We have therefore used Mozilla extensively as the test platform for the first generation Connexions system.

4.4 Displaying CNXML using Style Sheets

Since the stylistic information needed to display a document is not stored in the document itself, it must be contained in an external resource known as a style sheet. A style sheet simply contains a series of rules dictating how to display information marked up in a specific language. Browsers or other applications that wish to present the document to users in a particular fashion must retrieve the document and then apply the appropriate style sheet.

The W3C has specified two languages for use in constructing style sheets: Cascading Style Sheets (CSS) [24] and the Extensible Style Language for Transformations (XSLT) [25]. CSS has a syntax similar to declarations in the C programming language and is already widely used to format HTML pages. XSLT is a newer style sheet
language, written in XML. CSS applies stylistic information like fonts and spacing to the tags in a document, but XSLT allows the document itself to be transformed, perhaps into a different XML language. The two are not mutually exclusive, however, and are often used in conjunction. Connexions takes advantage of this fact to display CNXML on the web. We use an XSLT style sheet to transform CNXML into HTML, and then a CSS style sheet to present the specifics of how to display the HTML content. Figure 4.3 shows the results of applying two style sheets with different formatting styles to the example CNXML module in Figure 4.2.

Style sheets and XML are not confined to the web. There exist style sheets and languages which can transform XML into formats suitable for printing as well, such as PDF. This is an important factor for the Connexions project as well, since oftentimes students and professors alike may want to have access to a printed version of select materials.
Chapter 5

Component R: Module Repository

The module repository component has two primary functions. The first is to store the content modules (see Chapter 4), and the second is to provide access to the modules for those components that need it. For details about how each component communicates with the repository, refer to Sections 6.2, 7.3, and 8.2. The architecture for the repository is divided into two sections, one for each of its functions.

5.1 Module Storage

For the module storage function there are two issues to address. The first is where to physically store the modules, and the second is how to store the modules.

5.1.1 Where to store the modules

One possible solution would be to use a distributed storage system where different modules are stored and maintained at separate sites. This system presents several drawbacks, however. In a distributed system there are many sites which can fail, rather than just one. If one or more of the module storage sites goes down, links to those modules from surviving sites would be crippled. This could lead to a system where one or another portion of the module pool would randomly be inaccessible. For the Connexions system, this is unacceptable behavior. The content available to course instructors should not change based on the time of day. Another problem is that of maintenance. When a module changes it may be necessary to update other modules that link to and from it. This process could be difficult if the related modules are scattered across several sites.
In comparison, a central storage system is simple to maintain, as the modules all reside in one location on a single machine or cluster of machines. There is also no need to worry about parts of the module pool suddenly becoming unavailable. There are two primary concerns to a central storage system, but these are both easily addressable.

The first concern is that the central location may be too far (in terms of routing hops) from some users. This issue can be addressed by instituting a system of distributed proxy cache servers as shown in Figure 5.1. In this system the modules are stored centrally but are accessed by users via local Connexions proxies which cache commonly requested modules. The decision to implement such a system and where to locate the proxies should not be made until after the locations showing high concentrations of Connexions accesses can be determined.

![Figure 5.1: Distributed proxy cache system](image)

The second concern in using centralized storage is the large amount of disk space required to store modules. This is also not an immediate concern. In preliminary tests with over two hundred content modules, we found the average module size to be around 53kB (kilobytes). A twenty gigabyte hard drive, commonly available.
could store almost 400,000 such modules. Should the Connexions project exceed this magnitude, we will need to address this issue. At that size, it is likely that the module pool will tend to have clusters of modules around specific disciplines. One or more of these module clusters could be managed at a separate site with less risk of link breakage if a site goes down. In order to present a common front-end to users, the central Connexions system would maintain a module directory, redirecting clients to the correct site and allowing the storage implementation to be transparent to the user.

5.1.2 How to store the modules

One approach to storing modules would be to take advantage of the fact that content modules are XML documents (see Section 4.1) and are therefore ASCII text. This enables us to use version control software like the Revision Control System (RCS) [26] or the Concurrent Versions System (CVS) [27]. These systems retain the revision history of a document, allowing module authors to view previous versions and easily track or even revert changes to a document. One key limitation to such an approach is the difficulty of searching. Examining every module on the disk in sequence is not a scalable approach. On the other hand, modern database software is highly optimized for searching and data retrieval, but does not retain version information. To obtain the best of both worlds, the Connexions system combines these two approaches. The repository component stores the module text in a version control system to maintain the revision history and also stores the module metadata in a database for easy search and retrieval. The metadata contains module titles, keywords, and the linking structure. This indicates which modules are linked to which other modules, as well as the strength of the link.
5.2 Module Access

Access to the modules is handled separately from the module storage so that other Connexions components need not be aware of the specifics of the underlying storage implementation. The repository uses a web server to provide a uniform interface between the modules and the other components.

While the latest versions of modern web browsers do contain some support for displaying XML documents, the support is often rudimentary or incomplete. The web server must therefore perform some initial processing on the modules and present them to the client browser as HTML documents (to work with MathML, the Netscape 6 and Mozilla browsers require documents to be in the Extensible Hypertext Markup Language (XHTML) [28], a reformulation of HTML in XML). The flow of information in this case can be seen in Figure 5.2. The server uses an XSLT style sheet to transform the module from its storage format as a CNXML document into an HTML document. It then sends the HTML document along with an accompanying CSS style sheet to the browser.

![Diagram](image)

**Figure 5.2:** Server side module processing

The processing requirements for performing the XSLT transformation are not excessively large, but nor are they insignificant. DataPower Technology Inc. performed a recent preliminary benchmark [29] of XSLT processors. They found that on a five
hundred megahertz machine. James Clark's XT processor [30] had a throughput of about 230kB/s. While Microsoft's MSXML [31] processor performed faster, it is only available on Microsoft Windows platforms and is does not conform as well to the XSLT standard. With an average module size of 53kB, the web server would be able to process on average around four module requests per second before performance took a significant hit.

To ease the server's processing burden, we can trade disk space for processing power by pre-transforming the modules and storing the HTML output. Once the transformations have been performed initially, this solution virtually eliminates the extra processing time. It comes at the expense, of course, of needing to store two copies of every module. A more middle ground solution would be to use a caching mechanism, where the server retains transformed versions of only commonly requested modules.

The ultimate solution is simply a matter of time, however, as browser manufacturers improve their standards compliance. In particular, when browsers fully support on-the-fly XSLT transformations and the application of CSS to XML, the Connexions system will be able to switch to the data flow model pictured in Figure 5.3. Here the server simply passes the CNXML document and the XSLT and CSS style sheets directly to the client, which performs the necessary processing.

![Diagram of Client side module processing](image)

Figure 5.3: Client side module processing
Most accesses to the repository are read-only requests for modules or module metadata (e.g., link information). These requests can be handled anonymously through a web browser. The only component that requires write-access to the repository is the module authoring component, described in Chapter 6. An author who wishes to add new modules or edit existing ones must first obtain a username and password from the Connexions project. The repository stores these names and passwords and performs author authentication. To provide secure authentication, the web server uses the Secure Socket Layer (SSL) protocol [32].
Chapter 6

Component A: Content Authoring

This component consists of tools to facilitate the creation of content modules and their integration into the Connexions system. These tools perform two primary functions: assisting in module creation, and communicating with the repository.

6.1 Creating Content Modules

There already exists a wealth of educational material in the form of books, lectures, and web pages. The conversion of printed materials to electronic form is a topic of much research and is beyond the scope of the Connexions project. We will confine ourselves, therefore, to the conversion of existing electronic formats to CENXML and the creation of new CENXML content.

The primary difficulty in converting legacy content is the fact that most prior formats are primarily presentation-based, rather than content-based. Automated converters, therefore, must rely heavily on heuristics and are prone to misinterpreting the author's meaning. Current converters focus on syntactic transformation and rely on human intervention to handle the semantic content.

Much science and engineering content is in the \texttt{\LaTeX} format because of its facilities for handling mathematics. There are tools for converting from \texttt{\LaTeX} to XML and MathML, but they invariably use Presentation MathML rather than Content MathML. While Presentation MathML may display correctly under the Connexions system, it possesses none of the benefits of semantic content (see Section 4.1). Presentation MathML should therefore only be used as a temporary step between other math formats and Content MathML. For non-mathematical content, authors frequently use
word processing software such as Microsoft Word or Adobe FrameMaker [33]. Most such applications provide the ability to export documents to HTML, which the author can then manually edit to change the presentation-oriented HTML tags into content-oriented CNXML tags.

While there will always be legacy content, we must also ensure that module authors can easily create new content from scratch. Different content authors have widely differing tool preferences. The Connexions system could not possibly create a single editor to please everyone, therefore we will simply make recommendations and provide utilities to fill in the gaps in functionality.

There is a large range of software tools available for creating electronic materials. At one end of the spectrum is the simple text editor. Since XML files are human-readable text files, it will always be possible to use such a tool to create modules. Some editors even provide XML-friendly features such as automatic indentation and syntax highlighting. To assist authors who use text editors to create modules, Connexions provides utilities to verify that a module conforms to the CNXML specification [14] and to convert the module to HTML for viewing in a browser.

At the other end of the software spectrum are WYSIWYG ("what-you-see-is-what-you-get") applications. Most modern word processing software falls into this category. In these programs, the user makes changes to the document on-screen which is displayed in the same way as the printed version. The problem is that most WYSIWYG editors are also WYSIATI (“what-you-see-is-all-there-is”). They are primarily concerned with the appearance and formatting of a document, not its content and structure. This approach does not lend itself well to developing XML content.

There is still hope, however, for those users who prefer to use a WYSIWYG editor because it allows them to see how the final document will look. There exist specialized XML editors that provide pseudo-WYSIWYG behavior. These programs present multiple views to the user. One view shows the XML document structure
(usually as a tree), and the other view can be configured to display the result of applying a style sheet to the document. In this way the user can directly edit the document's content (not possible with true WYSIWYG editors) and still see how it will look.

Finally, when creating a module, whether from scratch or from existing material, authors must determine a suitable length for the module. To fully realize the potential of interconnections, modules should be small enough that they cannot be further subdivided. However, they must contain enough material to stand reasonably on their own. Put another way, modules should include just enough information such that they contain a reasonable treatment of a single topic, but no more. Content authors must use their judgment to decide how much material is necessary to create a sufficient standalone module.

6.2 Communicating with the Repository

To become part of the Connexions system a module must be submitted to the central repository. Communication with the repository does not require special software on the part of the author and may be done using any web browser that supports SSL for encrypted sessions. To login to the repository, an author must first obtain a username and password, as described in Section 5.2. Once authenticated, the author may edit existing modules or add new ones by using the appropriate web forms and uploading the module files. When creating new modules, content authors should search the repository for related modules. They may then create links to those other modules, assigning a link strength that depends on the depth of the relationship between the modules.
Chapter 7

Component C: Course Creation

Component C, the course creation component, allows instructors to use the Connexions system to build customized course materials. This component has three responsibilities: aiding the instructor in course construction, preparing the course for presentation, and communicating with the central module repository.

7.1 The Course Composer

The Course Composer is an application that enables instructors to author course materials using content modules as building blocks. This is done in three steps: constructing a course-path, adding annotations to modules, and selecting a style sheet.

First, instructors must select a sequence of modules from the repository to use in teaching a course. This provides the students in a course with a path through the sea of modules, while still allowing them to see how topics interrelate (see Figure 7.1). To find modules for inclusion in a course, instructors can search the module repository by author, keyword, title, or text string. The Course Composer provides the instructors with several options when constructing a course-path. If a course is longer than a few modules, the path can be segmented into chapters. The path can also fork to form side-paths for topics that some students may need but others may skip. Instructors can even create their own modules if they cannot find ones in the repository to suit their needs.

To enable instructors to tailor materials for their courses, the Course Composer provides a fine-grained annotation facility allowing annotations at the level of indi-
Figure 7.1: Module pool (a) and module pool with course-path (b)
vidual words. This allows instructors to add their own comments to the existing material, perhaps to call attention to an important point or to offer an alternative explanation if the one provided is unclear. Additionally, instructors may want to add transitional material between modules to improve the flow of the text.

Since annotations are specific to a course rather than any given module, they must be stored outside of the module repository. To accomplish this, the Course Composer makes uses of the metadata Resource Description Framework (RDF) [34] and XPointer [35], the W3C's language for pointing to parts or fragments of XML documents.

Finally, the Course Composer presents instructors with an opportunity to customize the "look and feel" of their course materials by selecting a style sheet. Since the modules contain only content information, they may be styled completely according to the tastes of the instructor. In particular, if the course modules contain a large amount of mathematics, the course instructor will want to create a style sheet to define his or her own notation (see Section 4.3). The Connexions system provides default style sheets for instructors who do not wish to design their own.

### 7.2 Preparing the Course for Presentation

Once instructors have created the course materials using Course Composer they will want to present them to the students. The Connexions system provides two ways to do this. Instructors may choose to use either one or both of the methods.

The first method of presenting the course is to store the necessary information on a web server local to the instructor's institution. This information is output by the Course Composer upon the successful creation of course and consists of the following: an XML file describing the course path, any local modules the instructor created, the module annotation file, and the selected style sheet for the course. Once these resources are available over the web, students may use the module navigation tools (see Chapter 8) to browse the course materials.
The second method of presenting a course is to generate a printed version. The Connexions system provides a utility for accomplishing this. In addition to the information from the Course Composer output, this tool requires the content modules selected by the instructor from the repository. It will assemble the modules, apply the instructors customizations, and convert the materials from XML into PDF for printing.

7.3 Communicating with the Repository

The Course Composer must communicate with the module repository so that instructors can search for modules appropriate for inclusion in their courses. This requires web access, although it is conceivable that once a list of modules has been downloaded, the list could be cached on the local machine for future use. Accessing the repository to search for and view modules does not require special permissions and can be performed with a web browser, although the Course Composer will have some search capabilities built in.
Chapter 8

Component N: Module Navigation

One of the Web’s defining characteristics is the ability to browse from page to page following a seemingly endless trail of links. Without a clear map it is quite easy to get lost. This is especially true in the Connexions system, which has an abundance of links. The goal of the module navigation component is to help users see where they are, where they were, and where they are going. This functionality is provided by the Connexions system’s Roadmap tool, either as a browser plug-in or as a complete customized Connexions browser using Mozilla’s Extensible User Interface Language (XUL) [36]. The two main functions of the Roadmap are to keep users apprised of their location and to communicate with other components in the Connexions system.

8.1 Keeping Users On-track

The Roadmap gives users a sense of how the current module they are reading relates to the modules around it. It also shows the previous and next modules in the course-path defined by an instructor using the Course Composer. It presents the connections between modules in a way that preserves the interconnected, non-linear quality of the module pool, but also highlights the instructor’s chosen path. This allows students to explore related topics and then easily return to the course-path.

To present students with information at multiple levels, the Roadmap contains three map views: course view, chapter view, and module view. The course view provides a high-level overview of the course, allowing students to see the chapters that are present and the overall path of the course. Chapter view allows students to zoom in on a single chapter, showing the individual modules that comprise the chapter
Figure 8.1: Prototype Roadmap implementation showing course view (left pane) and chapter view (right pane)

and links between them. Figure 8.1 contains a screen-shot of the Mozilla browser displaying prototype course and chapter views. Module view is the lowest level or most zoomed-in view. It displays information about a particular module including links to other modules not in the course. To maintain perspective in the course, it highlights the previous and next modules within the course context. Students may specify a minimum strength of link to control how many links module view displays. Figure 8.2 shows a prototype implementation of module view. Outside the context of a course the Roadmap will not display the course and chapter views since there is no defined course-path. The Roadmap will still present the module view to allow users to see links to related modules.

Once a student has selected a module for viewing, the browser displays that module using the instructor's specified style sheet and displays any annotations the in-
Figure 8.2: Prototype implementation of Roadmap module view

The instructor may have added. As the student navigates the course, either by clicking on one of the map views or by following a related link inside a module, the map views automatically update to reflect the student’s new location.

In addition to the three map views and the current module being displayed, the Roadmap presents users with the Connexions toolbar. This toolbar contains buttons for switching between map views, going forward and backward along the course-path, starting a new course, searching the module repository, and looking up selected words in an online dictionary.

8.2 Communicating with Other Components

The Roadmap must communicate with the module repository (see Chapter 5) to allow students to search for and display the various content modules. This is done anonymously over the web and does not require special permission to access the repository. In order to display a course path for students, the Roadmap must have access to the output of the Course Composer for that particular course. To begin using the Roadmap for a course, students must specify the web address of the server where the course instructor has stored those files (see Section 7.2). This allows the
Roadmap to display the course-path along with any annotations and local modules the instructor has defined.
Chapter 9

Conclusions and Future Work

The Connexions architecture has several benefits over traditional systems for handling educational materials. The Course Composer tool gives instructors enormous flexibility in selecting and customizing course materials by allowing them to select modules, sequence them, and add their own annotations. Instructors can tailor the look and feel of the content to their own preferences, even the mathematical notation. This is made possible through the Connexions system’s use of semantic markup to separate content from presentation.

By storing content as interlinked modules, Connexions also takes advantage of people’s natural ability to learn by forming connections. By displaying module connections, the Roadmap tool makes it easier for students to visualize concept relationships in their courses. It also shows them their location along the course-path, enabling them to see where they’ve been in relation to where they are heading.

These features make the Connexions system a powerful architecture for creating, storing, customizing, and browsing educational materials. The system is also quite flexible and easily extended. Future versions may incorporate features such as student annotations, collaborative editing, and module feedback forums.

Similar to the way instructors annotate the modules in a course, Connexions could allow students to save personal annotations to a module. This feature would allow students to highlight key concepts or save personal explanations. These annotations would need to be private and stored locally on the student’s computer. Implementing this feature would require an extension to the navigation component (see Chapter 8) to allow students to input and save the annotations, as well as a way to distinguish
student annotations from instructor annotations.

Facilities for collaborative editing could allow geographically separated contributors to work on the same module. This would require additional tools in the module authoring component (see Chapter 6) to enable communication with other authors and to allow authors to view module changes made by others. Using a version control system in the repository is a definite advantage here since authors could access previous versions to perform comparisons.

The Connexions system could also provide a feedback forum for each module so that students could communicate with the module's author and with each other. This would allow students to explain concepts to each other and could help create a sense of community around a course or discipline. In addition, the module author would gain a valuable source of feedback by reading students' comments on the module. To implement this we would need to address such issues as moderation, whether forums should be course-specific, and where to store the messages. The central repository (see Chapter 5) would require expansion if the messages are to be stored there. Similarly, if forums are course-specific the course creation component (see Chapter 7) would need to be enhanced.

The very nature of the Web is to evolve. It has developed at an extremely rapid pace and has given us access to an enormous wealth of knowledge. More importantly, it is changing the way we think about storing and disseminating information. To merely present digitized versions of the same old linear content would be to miss out on great potential benefits. The Connexions system takes advantage of this potential to provide instructors and students with a powerful new framework for education.
Bibliography


