AN EMBEDDED TREE DATA MODEL FOR WEB CONTENT ADAPTATION

by

YANMING WANG

B. Eng., Beijing Institute of Technology, 1997

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

We accept this thesis as conforming to the required standard

..........................................................
..........................................................
..........................................................
..........................................................
..........................................................

THE UNIVERSITY OF BRITISH COLUMBIA

October 2004

© Yanming Wang, 2004
ABSTRACT

This thesis presents a novel data model named Content Container Tree (CCT) to enable dynamic web content adaptation for heterogeneous terminal devices. The rationale for the CCT data model is that a web page can be visually divided into several segments and each segment can be iteratively divided into several smaller sub-segments until certain granularity level has been reached; the hierarchy of the web page segments can be expressed with a tree structure; thus, by traversing this segmentation tree, the user can find and locate a particular segment of interest which is at the dimension comparable to the screen size of the terminal device. The CCT data model organizes the semantics of the hierarchical content segments with a tree structure and embeds this semantic tree into the content DOM tree to enable the user to navigate through the content segments at different levels and retrieve the target content segment according to their semantics. Based on the CCT data model, the high level structural adaptation can be explicitly separated from the low level presentational adaptation, resulting in a highly modular and extensible framework for web content adaptation which is proposed in this thesis too. The CCT web content adaptation approach is distinguished from other approaches in the following aspects: first, the CCT approach is data oriented rather than process oriented; second, the CCT approach supports multilevel granularity of the contents; finally, CCT approach is dynamic; result pages can be generated on the fly by request. As an empirical evaluation, a content adaptation application is built according to the CCT adaptation framework to show the feasibility and effectiveness of our approach.
Table of Contents

ABSTRACT ........................................................................................................................................ II

TABLE OF CONTENTS ................................................................................................................... III

LIST OF TABLES ........................................................................................................................... VI

LIST OF FIGURES ........................................................................................................................ VII

LIST OF ABBREVIATIONS ............................................................................................................. IX

ACKNOWLEDGEMENTS ................................................................................................................ XI

CHAPTER 1: INTRODUCTION ...................................................................................................... 1

1.1 Motivation .............................................................................................................................. 1

1.2 Two Levels of Content Adaptation ...................................................................................... 2

1.3 Organization of This Thesis ................................................................................................ 5

CHAPTER 2: BACKGROUND ....................................................................................................... 6

2.1 Related Works ...................................................................................................................... 6

2.1.1 Server Side Approaches ................................................................................................. 6

2.1.2 Client Side Approaches ................................................................................................. 7

2.1.3 Intermediary Approaches .............................................................................................. 8

2.2 Enabling Technologies ........................................................................................................ 16

2.2.1 XML ........................................................................................................................... 16

2.2.2 XHTML ...................................................................................................................... 20

2.2.3 XSL and XSLT ........................................................................................................... 21

2.2.4 XPath ........................................................................................................................ 22
List of Tables

Table 2-1: An HTTP request containing both a header part and a body part. ............... 27
Table 2-2: An HTTP response containing both a header part and a body part............. 27
Table 4-1: Example granularity definition................................................................. 53
# List of Figures

Figure 2-1: Screenshot of same page with power browser using text summarization..... 10

Figure 2-2: Digestor web content adaptation framework. ........................................... 12

Figure 2-3: Two-level hierarchy with thumbnail view representation ...................... 14

Figure 2-4 WAP 2.0 programming model.................................................................... 25

Figure 2-5: WAP 2.0 optional proxy model ................................................................. 25

Figure 2-6: A transaction flows through a series of WBI MEGs ................................. 28

Figure 3-1: CCT web content adaptation framework .................................................... 31

Figure 3-2: A sample web page layout with multiple segments ................................. 37

Figure 3-3: CCT data model for the UBC home page ................................................. 42

Figure 3-4: The layered CCT data model ...................................................................... 46

Figure 4-1: Container element coding example ............................................................ 54

Figure 4-2: CCT container index user interface generation ........................................... 57

Figure 4-3: User interface for the UBC homepage ......................................................... 58

Figure 4-4: An XSLT script example for generating container index page ................. 60

Figure 4-5: A container index page generated with XSLT .......................................... 61

Figure 4-6: XSLT script for extracting content node ..................................................... 61

Figure 4-7: A content page generated with XSLT ......................................................... 62

Figure 4-8: <ct> tags embedded in HTML tags ............................................................. 63

Figure 5-1: ServerFileGenerator design ..................................................................... 67

Figure 5-2: CctXslProcEditor design ...................................................................... 68
Figure 5-3: XSLT style sheet template ................................................................. 70
Figure 5-4: Test bed configuration ................................................................. 72
Figure 5-5: The UBC homepage on the CCT adaptive web server .................. 76
Figure 5-6: Granularity hierarchy of the content in the right column container .... 77
Figure 5-7: The Yahoo homepage on the CCT adaptive web server ................. 79
List of Abbreviations

ASP     Active Server Page
AST     Abstract Syntax Tree
BMP     Bit Map
CC/PP   Composite Capability/Preference Profiles
CCT     Content Container Tree
CDPD    Cellular Digital Packet Data
CDMA    Code Division Multiple Access
CSS     Cascading Style Sheet
DOM     Document Object Model
GIF     Graphics Interchange Format
GSM     Global System for Mobile Communication
HTML    Hyper Text Markup Language
HTTP    Hyper Text Transfer Protocol
J2ME    Java 2 Mobile Edition
J2SE    Java 2 Standard Edition
JAXP    Java API for XML Processing
JPEG    Joint Photographic Experts Group
JRE     Java Runtime Environment
LAN     Local Area Network
MEG     Monitor Editor Generator
PC      Personal Computer
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>PHP</td>
<td>Personal Home Page</td>
</tr>
<tr>
<td>SAX</td>
<td>Simple API for XML Parsing</td>
</tr>
<tr>
<td>SGML</td>
<td>Standard Generalized Markup Language</td>
</tr>
<tr>
<td>STU</td>
<td>Semantic Text Unit</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VIPS</td>
<td>Vision-based Page Segmentation</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WAE</td>
<td>Wireless Application Environment</td>
</tr>
<tr>
<td>WAP</td>
<td>Wireless Application Protocol</td>
</tr>
<tr>
<td>WBI</td>
<td>Web Intermediaries</td>
</tr>
<tr>
<td>WML</td>
<td>Wireless Markup Language</td>
</tr>
<tr>
<td>XHMTL</td>
<td>Extensible Hypertext Markup Language</td>
</tr>
<tr>
<td>XPATH</td>
<td>XML Path Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSL</td>
<td>Extensible Style-sheet Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensible Style-sheet Language for Transformation</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I would take this opportunity to express my gratitude to my supervisors Dr. Victor Leung and Dr. Mathew Yedlin for their great supports and inputs to this research.

I would also give my sincere thanks to Dr. Philippe Kruchten and Dr. Lee Iverson for their very constructive suggestions and valuable opinions on this thesis.

Yanming Bruce Wang

The University of British Columbia
September, 2004
CHAPTER 1

INTRODUCTION

1.1 Motivation

The World Wide Web experienced an explosive growth in the last decade due to the wide spread of the Internet and the rapid increase in the number of the personal computers (PC). The next decade will see the booming of the mobile Internet which issues many new challenges in terms of devices, protocols, networks and user preferences to the web. One of these challenges is the diversity of the terminal devices in the input/output capability, processing capability and supported standards etc. To meet this challenge, a well-defined data model is needed for web content organization and presentation.

The existing HTML-based web content data model is mainly inherited from the traditional paper-based publication industry. It focuses on the layouts and visual effects rather than the semantics and the structures of the content. In the paper-based publications such as newspapers, magazines and books, the size of the paper for printing is pre-determined; similarly, a personal computer with a display resolution at least 800 x 600 pixels is usually assumed when most of today’s web pages are designed. Hyper Text Markup Language (HTML) is a language that mainly describes how content are displayed rather than what the contents are about and how they are related. Although Cascading Style Sheets (CSS) can be applied to bring more flexibility on the presentation of a web document, its main purpose is still defining different visual effects of the content.
rather than the semantics of the content. For this reason, there exists the incompatibility between the display-oriented web content data model and the capabilities of the heterogeneous terminal devices. And the content adaptation is needed to overcome the incompatibility to allow a web content to be accessed from heterogeneous terminal devices.

Content adaptation is about generating multiple versions of presentations for the contents according to the particular characteristics of the client device which is trying to access that content. However, the display-oriented web content data model of today makes it hard to dynamically modify the presentation of the contents because some information are conveyed implicitly by the layouts and the visual effects of the content display. Changing the presentation of the contents may cause losses of the information and yield illegible results. Thus, we want to design a data model which can overcome the shortcomings of the current HTML-based web content data model by including semantics into the data model. This data model should also be compatible with the HTML-based data model to enable dynamic adaptation of existing web pages.

1.2 Two Levels of Content Adaptation

The term “content adaptation” has two levels of meanings. The first level, which we call the low-level adaptation, is about the adaptation on the format or the encoding of the content. For instance, a web page is written in HTML, but the terminal device only understand Wireless Markup Language (WML), so it is necessary to translate the HTML syntax into the WML syntax. For another example, the content being requested is a 32 bit colors 800x600 pixels Bit Map (BMP) image, but the terminal device can only render
Graphics Interchange Format (GIF) images with 256 colors and no larger than 300x200 pixels, so converting the image from its original format, size and color depth into the destined format, size and color depth supported by the terminal device is necessary. The low-level adaptation only involves the encoding and the format of particular web content, it has little to do with the semantics of that content or its relationship with other contents, so it is the adaptation on the content data.

The second level, which we call the high-level adaptation, is about adaptations on the structural, navigational and functional aspects of the content. For instance, a typical web page which is designed for rendering on desktop computer is not suitable for a mobile phone’s screen; to fit it in, we need to divide it into several sub-pages, that is structural adaptation; and we need to design a scheme to let the user navigate through all sub pages, that is navigational adaptation; moreover, we might want to remove some contents that are not suitable for the terminal device such as advertisements, animations and embedded script codes etc., that is functional adaptation. To achieve the high-level adaptation, knowledge about the structures and functions of the target contents and sometimes the intention of the author is needed. So, the high-level adaptation is on the information about the content. However, as the primary purpose of HTML is to define the appearance rather than the structure and function of the contents, the high-level adaptation is more difficult than the low-level adaptation. There have been a few researches on abstracting the structural and functional information of a web page by analyzing its source code and the visual effects.

To manage both the information for high-level adaptation and the content data for low-level adaptation, we design a novel data model named the Content Container Tree
(CCT) data model. The CCT data model organizes both the high-level information and the low-level content data of a web page with an embedded hierarchical tree structure.

With the CCT data model, web content adaptation can be achieved in three steps. First, analyze and abstract the semantics and the structure of a web page to form the CCT. Second, navigate through the CCT and retrieve the target content data according to the user’s requirements. The first step and the second step together can be viewed as high level adaptation. The third step is to perform low level adaptation on the selected content data according to the terminal device’s capability. If the served document already conforms to the CCT data model, the first step can be omitted. In this case, the first step of adaptation is actually performed in the authoring stage. The advantages of CCT data model include:

- By applying the CCT data model, the high level adaptation and low level adaptation can be explicitly separated, thus making it easier to study and build web content adaptation systems.

- The CCT data model embeds the structural and functional information into a web page without requiring extensive changes to the existing HTML-based data model. So, an existing HTML document can be easily modified to conform to the CCT data model. Moreover, the current web architecture does not need any changes either.

- Besides content adaptation, CCT data model can also be utilized by other web applications such as searching engines to improve their efficiency and accuracy according to the high-level semantic information.
1.3 Organization of This Thesis

The rest of this thesis is organized as follows. In chapter 2, the related works and technology background on web content adaptation are discussed. In chapter 3, we present the CCT web content adaptation framework as well as the formal definition of the CCT data model. In chapter 4, we discuss the implementation of the CCT data model with XHTML in detail. In chapter 5, we present the design of the CCT adaptive web server application and the experiment results with this application. Finally, conclusions and future work are given in chapter 6.
CHAPTER 2

BACKGROUND

2.1 Related Works

To enable the heterogeneous terminal devices to access the web content, many efforts have been made and various solutions have been proposed. Among all these solutions, three general categories can be summarized, they are: server side approaches, client side approaches and intermediary approaches.

2.1.1 Server Side Approaches

Server side approaches include device-specific authoring and dynamic content generation. Device-specific authoring involves authoring a set of web pages for a particular type of terminal devices. For example, there exist web sites that are authored in WML for Wireless Application Protocol (WAP) compatible mobile phones. The advantages of this approach include the simplicity of implementation and high quality result. A content provider will simply create a set of new web pages based on the capabilities of a particular type of terminal devices. And since it is designed intentionally by human designers for the particular type of terminal device, the display effect is usually the best among all approaches. As the cons, the users of this particular type of devices will only have access to the web pages that are intentionally designed for them, thus limiting them from taking advantage of the huge number of other web pages on the web. On the other hand, due to the heterogeneous nature of the handheld terminal devices, the
content providers may have to provide many versions of web pages for the same content to fit various terminal devices. This will impose a heavy burden on the content providers to generate, maintain, synchronize and update their web sites.

Dynamic content generation systems generate different presentations from one common source content on the fly to fit various terminal device capabilities and user preferences. One example of such dynamic content generation system is Motorola Labs’ web-enabled museum system [1]. The system retrieves exhibit data from a content server based on a visitor’s context including location, information interests, and device capabilities etc. Motorola’s effort proved that dynamic content generation is feasible and can dynamically generate acceptable result for various terminal devices. However, its approach is very complex and requires a totally different architecture from the current file retrieval web server architecture, because all the relative information including the source content and the context information are stored in databases. Thus, it is still impossible for those legacy web sites to be accessed from the various terminal devices.

2.1.2 Client Side Approaches

Client side approaches are mainly about the techniques to provide the users the ability to interactively navigate a single web page by altering the portion of it that is displayed at any given time. Scroll bars, back and forward buttons are very common techniques falling in this category. More complex examples include the PAD++ system [2], which lets the user zoom and pan the device display over the document, and the Active Outlining [3], which lets the user dynamically expand and collapse sections of the document under their respective section headings. Client side navigation can work well if a good set of viewing techniques can be developed. But it requires the entire document to
be delivered to the client device at once which may waste valuable wireless bandwidth and memory. Besides, client-side navigation has high requirements on the hardware and software of the terminal devices which may not be met by many mobile terminal devices.

2.1.3 Intermediary Approaches

Intermediary approaches involve developing software that can take an arbitrary web document designed for the desktop environment, along with the characteristics of the target rendering device, and re-author the document through a series of adaptations so that it can be properly rendered on the device. This process is usually performed on a proxy, but it can also be performed either on the client side or on the server side. Examples of automatic content adaptation include the text summarization facility [4] developed by Buyukkokten et al. in Stanford University; the Digestor system [5] developed by Bickmore et al. in MIT; Browsing via thumbnail view [6] developed by Yu Chen et al. in Microsoft Research; Aurora system [7] developed by Anita Huang et al. in IBM Almaden Research Center.

Intermediary approaches do not require much extra changes on the content server side or the client side. The re-authored documents are usually light weight and require less bandwidth for transmission and less memory for storage than the original ones. So if an intermediary approach can produce acceptable results, which is legible, navigable and aesthetically pleasing, it will be the most preferred over other approaches.

In Stanford’s text summarization facility, an accordion representation is generated and the detail content can be folded or unfolded. The proxy begins by partitioning the page into ‘semantic textual units’ (STU). The result of this step is shown by the rectangles around sections of the page in Figure 2-1(a), which are not present in the
actual HTML. In summary, STUs are page fragments such as paragraphs, lists, or ALT tags that describe images. In a second step, the proxy uses font and other structural information to identify a hierarchy of STUs. For example, the elements within a list are considered to be item STUs nested within a list STU. Similarly, elements in a table, or frames on a page, are nested. Hiding the nested STUs finally completes summarization. Figure 2-1(b) exemplifies the result. Initially, each STU is represented by a single line on the personal digital assistant (PDA) screen. The arrows in Figure 2-1 denote the correspondence of STUs with their single-line representations. (The line numbers on the PDA are only for convenience of explanation and do not appear on the actual display).

The ‘+’ symbols next to some STU representations indicate the presence of hidden STUs at a deeper nesting level. Users may view STUs at those deeper levels by tapping their pen on the appropriate ‘+’ symbol, or by performing a left-to-right pen gesture. In response, the ‘+’ turns into a ‘−’ symbol, and the nested STUs are displayed indented. For example, the STU of line 4 in Figure 2-1(b) has been expanded, revealing lines 5–9. Then the STU of line 9 was expanded to reveal lines 10–13. The STU of line 3 has not been expanded, hence the ‘+’ on that line. Initially, only the top level of the STU hierarchy is shown on the screen. In Figure 2-1(b) this top level consists of four STUs in lines 1–4. Lines 5–13 were initially blank. Using this ‘accordion’ approach to Web browsing, users can initially get a good high-level overview of a Web page, and then “zoom into” the portions that are most relevant.
In our proposed CCT data model, we apply an ‘index’ approach which is similar to the ‘accordion’ approach of the Stanford’s text summarization facility to help the user navigate through a web page and locate the most relevant portion of the contents. The
main differences between the CCT data model and the text summarization facility lie in that:

1. The text summarization facility abstracts only the text content from a web page. Images and other HTML elements are abandoned. The CCT data model, on the other hand, preserves the full original HTML document. When a portion of the contents is selected, the respective segment of the web page is returned rather than just the text content.

2. The text summarization granularity is fixed. A summary line only corresponds to one certain text paragraph. The user can only choose to view the summary or the full text no matter what type of terminal device he is using. On the other hand, the CCT data model supports multilevel granularity which means the user can choose the granularity of the content to display according to the characteristics of his terminal device. Refer to the section 3.3 for detail explanations on multilevel granularity.

3. The text summarization facility applies static link anchors to organize the summary and its respective contents, while the CCT data model use the tree structure to define the relationships between the index and the contents, allowing dynamic operations for navigating and locating relevant content with tree-structure queries.

MIT’s Digestor system re-authors web documents through a series of transformations, and links the resulting individual pieces. The first thing that users of Digestor will typically do is to specify the size of display for their devices and indicate
the size of their default browser font; these are required in order to estimate the screen area requirements of the text blocks. Based on the estimation, Digestor will perform a series of transformation such as scaling down some images and replacing some texts with links etc.

The Digestor system and the CCT system both take into account of the heterogeneous terminal devices characteristics, thus they share a similar framework for content adaptation. As shown in figure 2-2, the Digestor system is implemented as a proxy server, which receives a request for an HTML document, retrieves the document from the specified HTTP server, parses the HTML and constructs an Abstract Syntax Tree (AST), labels each of the AST nodes with a unique identifier, and then retrieves any embedded images so that their size can be determined (as necessary). Once this has been accomplished, the planner is initialized with a state containing the AST for the original document. During each planning cycle, the planner selects the state with the best document version so far, then selects the best applicable transformation technique and applies it resulting in a new state and document version being generated.

![Figure 2-2: Digestor web content adaptation framework](image)

Figure 2-2: Digestor web content adaptation framework
The CCT content adaptation system can also be implemented as a proxy server. A Content Container Tree (CCT) is generated from the original HTML document. However, unlike the Digestor system which chooses a series of transformations to re-author the web page according to the terminal device’s screen size, the CCT system generate a multi-level indexes for the user to navigate through the web page, only the chosen portion of the contents will be further transformed on the fly according to the terminal device’s characteristics. Moreover, the Digestor system does not distinguish between navigational transforms such as outlining and content transforms such as image reduction; all transformation techniques are applied in one stage. In the CCT system, we explicitly defines two types of transformation, the high level transformation on the navigation and structure of the web page, and the low level transformation on the syntax, format and encoding of the contents in a web page. And these two types of transformation are accomplished in two separate stages in the CCT web content adaptation system. Please refer to section 3.1 for detail introductions to the CCT content adaptation framework.

Microsoft’s Thumbnail View system provides a graphical overview of the web page and allows the user to select a desired portion of the web page to zoom in for details. As shown in figure 2-3, a web-page is organized into a two level hierarchy with a thumbnail representation for providing a global view with each block of semantically related content represented by a different color. By tapping on a block in the thumbnail, a user can easily go to view the corresponding content which is formatted to fit well into a small screen.
The thumbnail view system best preserves the feel and look of the original web page. But its two-level hierarchy is insufficient for displaying large and complex web pages on a small screen terminal device. Moreover, its graphic-based navigation method constrains its applications to only those terminal devices which have good support to graphics. On the contrary, the CCT system adopts the more flexible multi-level indexes for navigation through a web page. The chosen contents can be displayed just as the original web page or can be further adapted according to the terminal device’s characteristics. The similarity between the thumbnail view system and the CCT system lays in that they both apply vision-based segmentation to re-organize the contents in a web page.

IBM’s Aurora system adapts web content based on semantic rather than syntactic constructs – facilitating navigation by streamlining the web interface according to abstract user goals. Instead of attempting to generalize for every web page, Aurora
analyzes web objects based on their semantic functions within aggregations of web pages. For example, within the context of a search-engine, it transforms a web object as a search-box rather than just an HTML form element. The goal is to make the abstract services provided by these collections accessible to the handheld device users. The advantage of Aurora is its ability to provide concept-oriented content adaptation. The trade-off is its scalability, since Aurora depends on a repository of custom content-extraction rules (one set of rules per web site). The generation and maintenance of these rules can be very labor-intensive and time-consuming. As a result, only a very small portion of web sites can be supported by Aurora.

The idea of adapting web contents based on semantics is also adopted by CCT, but for a different purpose. Unlike the Aurora system which works on the semantics to determine how to transform a particular web object according to its function in the context, the CCT system works on the semantics for content segments organization and navigation. In short, The CCT system utilizes the semantics of the hierarchical content segments to achieve high level adaptation on structures and navigations, while the Aurora utilizes the semantics of web objects to achieve low level adaptations on presentation and transcoding.

In conclusion, each of the approaches on server side, client side and intermediary has pros and cons. None of them can independently solve the problem of web content adaptation. The future architecture for web content adaptation is more likely to be a combination of more than one approach. First, on the server side, the content data model should be improved to be more structural and informative to allow intelligent processing. Second, on the client side, the terminal devices’ input, output and processing capability
should be improved to a certain level. A mono screen which can only display several lines of text will never provide satisfaction for web content browsing. With the rapid development in the personal handheld data device industry, colorful graphic screen with certain level of resolution will soon become the main stream, providing a base line for web content adaptation. Finally, the development of intermediary technologies will make the web smarter and more adaptable.

By comprehensively considering the various approaches for web content adaptation, the CCT data model overcomes the shortcomings of some existing attempts on web content adaptation and presents a flexible and extensible method to enable heterogeneous terminal devices to access the web contents.

2.2 Enabling Technologies

This section will introduce some state of the art technologies that can be applied to enable the web content adaptation.

2.2.1 XML

In some sense, the success of the World Wide Web attributes to the simplicity nature of HTML, the de facto standard markup language to author web documents. However, it is argued that the simplicity is also one of its weaknesses. The tags in HTML are mainly used for describing the visual presentation characters of a document, but have little to do with carrying the structural and semantic information about the content. For example, <H1> to <H6> tags are defined to emphasize the titles and subtitles of an essay, but in practice, text tagged by <H1> to <H6> are not necessarily a title, many people use
these tags only to get certain display effect on the font size or font type. Moreover, due to the loose grammar of HTML, it is hardly capable of advanced data manipulations and extractions. To overcome the shortcomings of HTML, XML (eXtensible Markup Language) [21] emerges in the mid 90’s as the standard for data representation and exchange on the Web. Note that XML is not a data model, but a meta-model to represent various documents. In a way, XML representations can be regarded as representations for semistructured data. Semistructured data is data that has some structure, but the structure may not be rigid, regular, or complete and generally the data does not conform to a fixed schema (sometimes the term schema-less or self-describing is used to describe such data). In semi-structured data, the information that is normally associated with a schema of the data is conveyed by the data itself. An example of semi-structured data is that text content could be represented as title, author, affiliation and paragraphs. Such data are not fully structured like relational structures, but they have partial structure. Semistructured data has gained the notice recently for the reason that it may be desirable to treat Web sources like databases which have no fixed schemas.

XML is a restricted version of SGML (Standard Generalized Markup Language), designed especially for Web documents. As a meta-language (a language for describing other languages), XML enables designers to create their own customized tags to provide functionality not available in HTML. And in this way, XML enables different kinds of data to be exchanged over the Web. The advantages of XML include: simplicity, open standard and platform/vendor-independent, extensibility, separation of content and presentation, support for the integration of data from multiple sources, ability to describe
data from a wide variety of applications, and more advanced search engines. The components of XML documents include the following:

- **Declaration:** the XML declaration defines the XML version and the character encoding used in the document. For example, `<?xml version="1.0" encoding="ISO-8859-1"?>` means the document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set.

- **Elements:** elements, or tags, are the most common form of markup. The first element must be a root element. An XML document must have one root element. All other elements must be within this root element. All elements can have sub elements (child elements). Sub elements must be correctly nested within their parent element. For example,

```xml
<root>
  <child>
    <sub child>…</sub child>
  </child>
</root>
```

XML Elements are extensible and they have relationships. XML documents can be extended to carry more information. Elements are related as parents and children.

- **Attributes:** attributes are name-value pairs that contain descriptive information about an element. The attribute is placed inside the start-tag after the
corresponding element name with the attribute value enclosed in quotes. For example, 
<file type="text">.

- Entity References: entity reference is used in an XML document to refer to an entity. An entity reference starts with an ampersand (&) and ends with a semicolon (;), for example, &lt; refers to the left angle bracket (<) which is a reserved character.

- Comments: The syntax for comments in XML is similar to that of HTML. For example, "<!-- This is a comment -->."

- CDATA sections and processing instructions: a CDATA section instructs the XML processor to ignore markup characters and pass the enclosed text directly to the application without interpretation. A processing instruction is of the form <?name pidata?>, where name identifies the processing instruction to the application. Since the instructions are application specific, an XML document may have multiple processing instructions that tell different applications to do similar things, but perhaps in different ways.

Also note that, with XML, elements are ordered, which means two XML segments with same elements but appear in different order are different. In contrast, attributes in XML are not ordered, thus it does not matter which attribute goes first in an element.

One of the most outstanding characteristics of XML is the separation of content and presentation. Unlike HTML, XML simply structures a collection of data to store in a text format document. It does not deal with how the data looks. Another significant
characteristic of XML is that every XML document can be expressed as a well-formed tree, making it easy to process with the XML parsers.

Since many other technologies including XHTML, XSLT and XPath are based on XML; XML is the most basic enabling technology for our research.

2.2.2 XHTML

XHTML (eXtensible HTML) [24] 1.0 is a reformulation of HTML 4.01 in XML 1.0 and it is intended to be the next generation of HTML. It is basically a stricter and cleaner version of HTML. The most differences between XHTML and HTML are:

- XHTML elements must be properly nested
- XHTML documents must be well-formed
- Tag names must be in lowercase
- All XHTML elements must be closed

XHTML applies a method called modularization to achieve user-agent interoperability. An XHTML module is a set of existing HTML elements (or tags) or further elements which offer a specific functionality. The modules can be combined with each other and with the desired feature sets of different user agents to form different XHTML subsets or different XHTML-conforming document types, or XHTML-family markup languages. The modularization of XHTML facilitates the creation of new markup languages. A certain XHTML-family markup language, a collection of XHTML modules, is used to encode the Web content so as to best fit the capabilities of a certain type of user-agent. Every such language must include the core modules, and may also use other XHTML-provided modules, other World Wide Web Consortium (W3C) defined modules,
or any other module that is correctly defined. One example of an XHTML-family markup language is “XHTML Basic,” which includes the minimal set of XHTML modules and some other modules such as Images, Basic Forms, Basic Tables, and Objects. It is designed for thin clients that do not support the full set of XHTML features. Another example of an XHTML-family markup language is XHTML Mobile Profile (XHTML MP). XHTML MP intends to provide a content authoring language suitable for resource-constrained devices such as mobile phones. It extends XHTML Basic by adding some presentation elements and support for internal style sheets. XHTML MP has been taken by WAP forum as its client-specific markup language in the latest WAP 2.0 Specification.

Because XHTML are linked to both HTML and XML, we make use of it to implement the proposed CCT data model in our research. Such XHTML implementation will maintain the basic structure and functions of a conventional HTML web document while bringing in the data processing capabilities of XML. Please refer to section 4.2 for detail discussions on how to utilize XHTML for implementing the CCT data model.

2.2.3 XSL and XSLT

In HTML, default styling is built into browsers because the tag set for HTML is predefined and fixed. The CSS (Cascading Stylesheet Specification) allows the developer to provide an alternative rendering for the tags. Similarly, Extensible Stylesheet Language (XSL) [23] is the standard style definition language for XML. XSL is a formal W3C recommendation that has been created specifically to define how an XML document’s data is rendered and to define how one XML document can be transformed into another document. XSL is the counterpart in XML of CSS but more powerful.
Extensible Stylesheet Language for Transformations (XSLT) forms a subset of XSL. It is a language in both the markup and the programming sense in that it provides a mechanism to transform a XML structure into another XML structure. While it can be used to create the display output of a Web page, XSLT’s main ability is to change the underlying structures rather than simply the media representations of those structures, as is the case with CSS. XSLT is important because it provides a mechanism for dynamically changing the view of a document and for filtering data.

In our research, we make use of XSLT to extract the proper content from a CCT document to dynamically generate and format a web page. Please refer to the section 4.5 for detail discussions and examples on how to use XSLT.

2.2.4 XPath

XML Path Language (XPath) [24] is a declarative query language for XML that provides a simple syntax for addressing parts of an XML document. It was designed for use within XSLT (for pattern matching). With XPath, collections of elements can be retrieved by specifying a directory-like path expression with predicates. XPath uses a compact, string-based syntax, rather than the XML element-based syntax.

XPath treats an XML document as a logical (ordered) tree with nodes for each element, attribute, text, processing instruction, comment, namespace, and root. The basis of the addressing mechanism is the context node (a starting point) and the location path, which describes a path from one point in an XML document to another, providing a way for the items in an XML document to be addressed. XPath can be used to specify an absolute location or a relative location. A location path is composed of a series of steps joined with ‘/’, which serves much the same function as ‘/’ in a directory path. Each ‘/’
moves down the tree from the preceding step. An XSLT operation can take the XPath expression as an argument. For example, `<xsl:copy-of select="/catalog/cd" />` selects cd node from the catalog. XPath also defines a set of standard functions for working with strings, numbers, and Boolean expressions. For example, the XPath expression “/catalog/cd[price>10.80]” selects all the cd elements that have a price element with a value larger than 10.80.

In our research, we make use of XPath to locate and select particular contents or information from a CCT document. Please refer to the section 4.5 for detail discussions and examples on how to use XPath together with XSLT.

### 2.2.5 JAXP

Many web developers have come to the conclusion that XML and Java programming language form the perfect pair because they complement each other so well. XML contributes platform-independent data -- portable documents and data. Java contributes platform-independent processing -- portable object-oriented software solutions. Thanks to the combination of XML and Java, many applications that cannot be accomplished within the limitations of HTML are now solvable. Although there are many XML tools and libraries based on other languages such as Python, Perl, and C, the majority of XML development is focused on Java, which is emerging as the language of choice for processing XML.

Java API for XML Processing (JAXP) [25] makes it easy to process XML data using applications written in the Java programming language. JAXP leverages the parser standards SAX (Simple API for XML Parsing) and DOM (Document Object Model) so that developers can choose to parse their data as a stream of events or to build a tree-
structured representation of it. The latest versions of JAXP also support the XSLT (XML Stylesheet Language Transformations) standard, giving control over the presentation of the data and enabling converting the data to other XML documents or to other formats, such as HTML. JAXP also provides namespace support.

In our research, we use JAXP along with XML, XHTML, XPath and XSLT to build applications that implement the proposed CCT data model for web content adaptation. Please refer to the section 5.1 for how these technologies can be integrated together to create a concrete application.

2.2.6 WAP

WAP (Wireless Application Protocol) [26] is an open specification developed by the WAP forum to empower mobile users to access the Internet via wireless communication networks. WAP works with most wireless networks such as CDPD, GSM, GPRS, CDMA, TDMA, and 3G. What WAP provides is an application environment and a suite of communication protocols for mobile devices to access the Internet and advanced telephony service.

The latest version of WAP is WAP 2.0 which adopts more existing Internet and Web technologies. WAP 2.0 incorporates HTTP and TCP into its communication protocol stack and migrates its content encoding language from WML to XHTML. The resulting WAP programming model is more closely aligned with the Web programming model. In the earlier versions of WAP, namely WAP 1.0 and WAP 1.1, a WAP proxy (often referred to as a WAP gateway) was required to handle the protocol inter-working between the client and the origin server. The WAP proxy communicated with the client using the WAP protocols that are based largely on Internet communication protocols, and
it communicated with the origin server using the standard Internet protocols. WAP 2.0 does not require a WAP proxy, since the communication between the client and the origin server can be conducted using standard Internet protocol HTTP/1.1 over TCP/IP. Nevertheless, although the deploying of a WAP proxy is no longer mandatory in WAP 2.0, it can optimize the communications process and may offer mobile service enhancements. The WAP 2.0 programming model is shown in figure 2-4. The WAP Optional Proxy Model is shown in figure 2-5.

Figure 2-4 WAP 2.0 programming model

Figure 2-5: WAP 2.0 optional proxy model
In our research, we’ll use the WAP 2.0 programming model and optional proxy model as the basic model for mobile handheld devices to access the adaptive web content. We’ll also conduct our experiments with WAP 2.0 compliant simulators. Please refer to the section 5.3 for details.

2.2.7 WBI

IBM’s WBI [17] is a programmable web proxy and web server. ‘WBI’ stands for Web Intermediaries. The WBI Development Kit provides a set of convenient and flexible APIs for programming intermediaries on the web. In this case, intermediaries are computational entities that lie along the HTTP stream and are programmed to tailor, customize, personalize, or otherwise enhance data as they flow along the stream.

WBI has a data model and a processing model. Briefly, the data model describes the way requests and responses are accessed and manipulated. The processing model describes (a) the way different modules within WBI are activated when requests are received from the browser or other client, and (b) the way these different modules work together to produce the response that is sent back to the browser.

WBI's data model is based on the request/response structure of HTTP version 1.0. Each request and each response consists of a structured part and a stream part. The structured part corresponds to the header and the stream part corresponds to the body.

Table 2-1 shows an HTTP request that contains both a header and a body. When WBI receives this request, it is parsed into these two parts. The header information is stored in an object of class DocumentInfo. The body information is made available through an object of class MegInputStream. The body information can then be read from the MegInputStream using its read(…) method. This technique is used because the body
portion of a request or a response can be very long and it is not practical to store it in memory as a string.

<table>
<thead>
<tr>
<th>HEADER (DocumentInfo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST <a href="http://www.ibm.com/java">http://www.ibm.com/java</a> HTTP/1.0</td>
</tr>
<tr>
<td>User-agent: Mozilla/4.0</td>
</tr>
<tr>
<td>Accept: text/html</td>
</tr>
<tr>
<td>Content-length: 15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BODY (MegInputStream)</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is a sample http request.</td>
</tr>
</tbody>
</table>

Table 2-1: An HTTP request containing both a header (structured) part and a body (stream) part.

<table>
<thead>
<tr>
<th>HEADER (DocumentInfo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP/1.0 200 Ok</td>
</tr>
<tr>
<td>Server: MyWebServer</td>
</tr>
<tr>
<td>Content-type: text/html</td>
</tr>
<tr>
<td>Content-length: 36</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BODY (MegInputStream)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;html&gt;</td>
</tr>
<tr>
<td>&lt;h1&gt;Hello, world&lt;/h1&gt;</td>
</tr>
<tr>
<td>&lt;/html&gt;</td>
</tr>
</tbody>
</table>

Table 2-2: An HTTP response containing both a header (structured) part and a body (stream) part.

Table 2-2 shows a typical HTTP response. When WBI receives this response, it is parsed in the same way as a request. The header information is stored in a DocumentInfo object and the body is made available through a MegInputStream object.

To produce new requests and responses, a WBI module is given a DocumentInfo object and a MegOutputStream object to manipulate. Simply set a property of the DocumentInfo object using either the setRequestHeader(…) or setResponseHeader(…) methods. When the header information has been set appropriately, the WBI module may begin writing the body content to the MegOutputStream using its write(…) methods.

In summary, WBI divides both HTTP requests and HTTP responses into two parts: the structured header portion and the unstructured body portion. The header is
manipulated through a *DocumentInfo* object and the body is manipulated through *MegInputStream* and *MegOutputStream* objects.

WBI’s processing model consists of four types of processing entities: the request editor, the generator, the document editor and the monitor. WBI is a programmable HTTP request and response processor. It receives an HTTP request from a client, such as a web browser, and produces an HTTP response that is returned to the client. The processing that happens in between is controlled by the modules programmed into WBI.

Figure 2-6 shows the flow through a typical WBI transaction. It goes through three basic stages: request editors, generators, and editors (which would be more appropriately called "document editors"). Request Editors receive a request and have the freedom to modify the request before passing it along. Generators receive a request and produce a corresponding response (i.e., a document). Editors receive a response and have the freedom to modify the response before passing it along. When all the steps are completed, the response is sent to the originating client. A fourth type of element, the Monitor, can be designated to receive a copy of the request and response but cannot otherwise modify the data flow. The Monitor, Editor, Request Editor and Generator modules are collectively referred to as MEGs.
WBI dynamically constructs a data path through the various MEGs for each transaction. To configure the route for a particular request, WBI has a rule and a priority number associated with each MEG. The rule specifies a boolean condition that indicates whether the MEG should be involved in a transaction. The boolean condition may test any aspect of the request header or response header, including the URL, content-type, client address, server name, etc. Priority numbers are used to order the MEGs whose rules are satisfied by a given request/response. When it receives a request, WBI follows these steps:

1. The original request is compared with the rules for all Request Editors. The Request Editors whose rule conditions are satisfied by the request are allowed to edit the request in priority order.

2. The request that results from this Request Editor chain is compared with the rules for all Generators. The request is sent to the highest priority Generator whose rule is satisfied. If that Generator rejects the request, subsequent valid Generators are called in priority order until one produces a document.

3. The request and response are used to determine which Editors and Monitors should see the document on its way back to the client. The document is modified by each Editor whose rule condition is satisfied, in priority order. Monitors are also configured to monitor the document either (a) as it is produced from the generator, (b) as it is delivered back to the client, or (c) after a particular Editor.

4. Finally, the response is delivered to the requester.

A WBI application is usually composed of a number of MEGs that operate together to produce a new function. Such a group of MEGs is a WBI plugin. Plugins
define the basic unit for installation, configuration, and enable/disable. A plugin bears
some resemblance to a Java Applet. It is a class that WBI instantiates at start-up, similar
to how a browser loads an applet. In our research, we develop our CCT application as a
WBI plugin. For detail design issues, please refer to the section 5.1 “CCT adaptive web
server design” for detail introduction on developing the application with WBI.
CHAPTER 3
CCT DATA MODEL

3.1 The Web Content Adaptation Framework

To apply the CCT data model for dynamic web content adaptation, we propose a proxy-based framework as shown in figure 3-1.

![Figure 3-1: CCT web content adaptation framework](image)

This framework mainly consists of three functional modules, the HTTP proxy module, the content reformation module and the adaptive transcoding module.

1. The HTTP proxy module receives the HTTP requests from the client, interpret the request and forward it to the web server. This module is also responsible for analyzing the HTTP request for information about the client and the target network characters. Based on the request and the information, the control logic will coordinate the work of the other two functional modules.
2. The content reformation module is an intelligent agent responsible for analyzing the web page structure from the original web server and changing that web page to comply with the CCT data model. The resulting intermediate CCT document will be saved in a cache for further consumption by the adaptive transcoding module. In the case that the original web page already conforms to the CCT data model, this module can be omitted, and the system becomes an adaptive content generation system.

3. The adaptive transcoding module makes queries to the CCT document in the cache according to the HTTP request. Based on the client information gathered by the HTTP proxy module, the adaptive transcoding module makes respective low level content transcoding to the query results from the CCT document to fit the terminal devices’ capability requirements.

Comparing this CCT web content adaptation framework with the Digestor web content adaptation system as shown in figure 2-2, we can see some similarities between the two frameworks. First, the two frameworks both have proxy modules which are used to accept and forward HTTP requests. Second, they both have intelligent modules to re-author a web page. Finally, they both use caches to temporarily store re-authoring results for further consumption by the user. The major difference between the two frameworks is that the Digestor system perform all the transformations in one step and store the final transformation results in the cache for the user to access sequentially; while the CCT system performs the transformation in two separate steps, the first step is to re-author the web document into an intermediate format and store it in the cache, the second step is to
extract a particular portion of the contents from the intermediate document and perform content transcoding on the fly according to the terminal device’s characteristics. The first step of transformation is accomplished by the content reformation module while the second step of transformation is accomplished by the adaptive transcoding module.

The main advantage of the CCT web content adaptation framework over the Digestor system and other frameworks for web content adaptation is that it explicitly separates the high level adaptation and the low level adaptation, where the content reformation module is mainly responsible for the high level adaptation by analyzing the semantics and the structure of the original web page and adding these information to the intermediate CCT document. The adaptive transcoding module is mainly responsible for the low level adaptation such as markup language transcoding and image reduction on the chosen portion of the content. The adaptive transcoding module can be viewed as a miniature of a content adaptation system such as Digestor except that it only deals with portions of content that are at comparable size to the terminal device’s screen. For this reason, it is much easier to determine the proper transformation techniques because those high level navigational and structural transformation techniques such as outlining can be omitted.

By separating the high level adaptation and the low level adaptation, the complex problem of web content adaptation can be reduced to two smaller sub-problems: finding algorithms to analyze the semantics and structures of a web page; and finding techniques to transcoding various types of contents.

The pivot for connecting the whole content adaptation system is the intermediate document which is the output of the content reformation module and the input of the
adaptive module. Thus, the data model for this intermediate document should be carefully designed to be able to carry the necessary high level semantics and structural information acquired by the content reformation module. The intermediate data model should also possess a structure that allow the adaptive transcoding module easily and efficiently extract both the high level information and the content data from it. To achieve such a data model, both the display-oriented nature of the HTML web pages and the heterogeneity nature of the terminal devices should be considered. In the following sections in this chapter, we’ll discuss the design of such a data model in detail.

3.2 Multilevel Web Page Segmentation Rationale

Microsoft’s thumbnail view browsing system as introduced in the section 2.1.3 inspired us that if we can segment a large web page into several small sub-pages at a size that is comparable to the screen size of the terminal device, these small sub-pages will be much easier for users to browse. Then, the process of web content adaptation can be divided into several steps:

1. Find a proper set of sub-pages that are either visually or logically independent. The granularity of these sub-pages should be decided according to the screen size of the terminal device.

2. Find an efficient method to allow the user to navigate through the sub-pages and pick the most relevant one.

3. Find a set of transcoding techniques to further transform the chosen sub-page to make it more suitable for being rendered on the terminal device.
Microsoft’s thumbnail view browsing system applies a two-level hierarchy to manage the sub-pages and a graphical overview to enable the user to navigate through the sub-pages. The limitation of such design is that with the two-level hierarchy structure, only one granularity level of the sub-pages can be defined, thus only those terminals with certain screen size are supported. The graphical overview navigation system also limits its usage to only those terminal devices with certain graphical display capability. Because the thumbnail view browsing system supposes the terminal devices are capable to display graphical HTML pages, no further transformation is made to the chosen sub-page.

To enable heterogeneous terminal devices to access the web contents, we need a more flexible design of the data structure to organize the multi-level segmentations of a web page, so that we can have sub-pages at various granularity levels for various screen size of the terminal devices. We also need a method to help the user to traverse this multilevel hierarchy to find the target sub-page of proper granularity. Based on this rationale, we design the CCT data model which mainly describes how multilevel segmentations of a web page should be organized and how to traverse the multilevel hierarchy to locate and retrieve the relevant information and data.

Before we introduce the detail design of the CCT data model, we first study an example of web page segmentation to gain some ideas about the vision-based segmentation method, principle and process.

Because HTML web page design are focused on the design of visual effects and the content layout, the semantics and structural information of the web page are usually conveyed implicitly by the visual effects of the content and the page layout. For example, the content in large or bold font is usually more important than those in small or light font;
the content in the central area of the page is usually more important than those in the edge
areas; logically related contents are usually geographically adjacent; logically
independent contents are usually separated by visual cues such as blank areas, vertical or
horizontal lines, or by applying different visual patterns including fonts, colors etc. As an
example, figure 3-2 shows the home page of UBC, which has a very typical layout design
of today’s web pages. In figure 3-2 we can see that the web page can be visually divided
into five main segments which are logically independent with each other. They are:

- The header bar, which contains the logo, the organization name and some
  navigational links.
- The left column, which is mainly used to display navigational links and indexes.
- The main content area, which is used to display the main content of the web page.
- The right column, which is used to display less important or irrelevant content.
- The footer bar, which is used to provide complementary information about the
  web page such as address, phone numbers, copyright and some additional links.

In addition, the main content segment and the right column segment can be
further divided into several sub-segments. This segmentation process can go on
iteratively until the finest granularity level by definition is reached.
Figure 3-2: A sample web page layout with multiple segments

Automatic web page segmentation can be achieved with algorithms such as Vision-based Page Segmentation (VIPS) [11-13]. VIPS makes full use of page layout features such as font, color and size. It first extracts all the suitable nodes from the HTML DOM tree, and then finds the separators between these nodes. Here, separators denote the horizontal or vertical lines in a web page that visually do not cross any node. Based on these separators, the semantic tree of the web page is constructed. A value called degree of coherence (DOC) is assigned for each node to indicate how coherent it is. Consequently, VIPS can efficiently keep related content together while separate semantically different blocks from each other. The granularity of segmentation in VIPS is
controlled by a predefined degree of coherence (PDOC), which plays a role as a threshold of the most appropriate granularity for different applications. The segmentation only stops when the DOCs of all blocks are no smaller than the PDOCs. For detail description of the VIPS algorithm, please refer to [12].

The display-oriented web page design works well for human readers facilitated with large screen devices, because human viewers can easily find out the patterns and logical relationships hidden behind the graphical layout. However, it is hard for the machines to recognize the patterns and understand the information conveyed by the graphical layout. It is also unsuitable for human viewers with small screen devices. Because the web page cannot be fit within one screen, the viewer will have to scroll both horizontally and vertically to get the whole picture of the page, making the discovery of the patterns difficult. Moreover, in some situations, the constraints on the display of handheld devices (such as color, images and font sets etc.) can even make it impossible to render a web page properly on the handheld device. For this reason, it is necessary to explicitly express the semantics and relations of the segments of a web page to support small screen devices and machine processing. The segmentation semantics can be embedded in the content by the author or be explored by an intelligent agent program with segmentation algorithms such as VIPS. With the segmentation semantics, the user can navigate through the segments hierarchy in a web page and choose the target segment of content at proper granularity to display. By this way, the high level adaptation on in-page navigations is achieved.
3.3 The CCT Data Model

To manage the multilevel segmentation hierarchy of a web page, we design a special tree-structured data model named the Content Container Tree (CCT) data model. Unlike the thumbnail view browsing system which uses flat sub-pages, the CCT introduces the concept of ‘container’ and ‘content’. The ‘content’ is simply a piece of HTML code that corresponds to a particular segment in a web page. The ‘container’ is like a box to hold the ‘content’. The CCT data model works in a way similar to how we organize stuffs when we move our home. When we move, we put a category of stuffs into a box and put a label on the box to tell what is in it, for example we may put all of our clothes in a box with a label “clothes” and put all of our shoes in another box with a label “shoes”. Then for the ease of transportation, we may use larger boxes to hold close categories of stuffs. For example, we may use a large box to hold the clothes box and the shoes box and the quilt box. And we may use another large box to hold the kitchen utensil box and the garage tool box. We may also want to apply labels such as ‘bedroom stuff’ and ‘kitchen and garage stuff’ on these large boxes to give us some idea of what kind of stuffs are in these boxes. Finally, we put all the boxes in a vehicle for delivering to the destination. The advantage of such a hierarchical organization is that we can quickly locate an article by inspecting the labels of the boxes without going through the detail stuff in each box. For example, if we want to find a pan for cooking, we know it must be in the ‘kitchen and garage’ box. After we open this box and find there are two small boxes ‘kitchen utensils’ and ‘garage tools’, we know it must be in the ‘kitchen utensils’ box. Then we can open the kitchen utensil box to dig out the pan we need.
The idea of the CCT ‘container’ is very similar to the moving box concept but with some differences. When we say “open a container”, we mean something different than opening a moving box. We use the term “expanding a container” to express the similar meaning of opening a box. We’ll explain this difference in detail in the following parts in this section. For now, we just want to show the basic idea of how the CCT containers work with the moving-box analogy.

Similar to the moving box, a CCT container is used to hold a piece of HTML code that represents a particular segment in a web page. A CCT container can also be used to hold other CCT containers to form a hierarchy, and this CCT containers hierarchy reflects the multi-level segmentation hierarchy we described in the forgoing section. That is, an upper-level container corresponds to an upper-level segment and a lower-level container corresponds to a lower-level segment. The advantage of this “content and container” design is that it does not require opening the detail contents to know what the content is about. To locate target content, we only need to traverse the container hierarchy and check the labels of the container. After we find the right container which contains the target content, we can open that container and retrieve the content just like what we do to find an article in the moving boxes.

The hierarchical structure of the containers and contents can be expressed as a tree. The root node of the tree stands for the upper-most container which contains the whole page; inner nodes are the top level coarser containers; and all leaf nodes are flat segmentation of the web page content. We refer the leaf nodes as the content nodes; we refer all the non-leaf nodes, which have at least one child node, as the container nodes. The container nodes may have one to many children nodes. These children nodes can be
either other container nodes or content nodes. A container conserves semantics information about the content segment it corresponds to.

Applying the description method of document representation in [14], the basic model of CCT is described as follows. A web page (root container) $\omega$ can be represented as a triple $\omega = \{\Omega, \Phi, \Delta\}$, where $\Omega = \{\omega_1, \omega_2, ..., \omega_N\}$ is a finite set of sub-containers. All these sub-containers are not overlapped. Each container can be recursively viewed as a sub-page with structures induced from the whole page structure. $\Phi = \{\phi_1, \phi_2, ..., \phi_T\}$ is a finite set of flat content segments. A container may only contain content nodes, where $\Omega$=NULL; or it may only contain sub-container nodes, where $\Phi$=NULL. In the case where neither $\Omega$ nor $\Phi$ is NULL, it means the current container contains some content that are not in any of its sub-containers. This is possible when some content is meaningful only in the upper-level segment, for instance a horizontal line that visually separates two sub-segments of the upper-level segment. $\Delta = \{\delta_1, \delta_2, ..., \delta_M\}$ is a set of rules that defines the relationship among the sub-containers of the current node; for instance $\delta_1$ may define a rule that “sub-container $\omega_1$ precedes all other sub-containers”. Figure 3-3 shows the diagram of the CCT data model for the UBC home page as presented in figure 3-2. Note that the circles in the diagram stand for the content nodes, while the rectangles stand for the container nodes. The hierarchy of the web page segmentation is reflected by the tree structure of the container nodes. The data model in figure 3-3 shows that the document container consists of five sub-containers: the header-bar container, the left-column container, the main-area container, the right-column container and the footer-bar container. The main-area container further consists of four sub-containers; each of which contains a text paragraph. The right-column container consists of three sub-containers too.
Such a tree structure of the containers can help the user to navigate through the content segments from a small screen terminal device.

The granularity of the segmentation is decided based on the size of the smallest screen supported, which means the iteration process of segmentation can stop when the size of the content segment is comparable to the smallest screen size supported by design. However, a user with large screen device does not have to go through all the segmentation hierarchy and display content segments at the finest granularity level. In fact, a user can choose any content segment which is at a granularity level higher or comparable to the screen size of his device. For instance, instead of displaying the three articles in the main area one by one, all the contents in the main area segment can be displayed together in one page. In the extreme case, the whole web page can be displayed in one page if the terminal device is a personal computer. This flexible-granularity feature

Figure 3-3: CCT data model for the UBC home page.
is also a significant advantage of the CCT data model over other web content adaptation approaches.

Now we can explain the difference between the CCT container and its moving-box analog. Due to the physical constraints, when we open an outer moving box, we can only see the inner boxes. If we want get some stuff out, we must open all the boxes one by one on the hierarchy until we reach the inner-most box which holds the thing we want. However, such physical constraint does not apply to the CCT containers. When we ‘open’ a CCT container, what we actually mean is to retrieve the content segment this container corresponds to no matter how many sub-level containers it has. So we define the ‘opening’ operation in CCT as retrieving the content segment the container corresponds to which includes all the content nodes of its own as well as its descendent containers.

To allow the user to inspect the sub-containers of the current container, we define another operation “expanding”. ‘Expanding’ a container means to read the labels of its immediate sub-containers. The ‘expanding’ operation is similar to opening an outer level moving box to inspect what inner boxes are in it.

Not all containers are eligible to both the “opening” and the “expanding” operations. For instance, the containers which contain only content nodes are not ‘expandable’. In what condition a container is ‘openable’ or not is a little bit more complex. Basically, all the containers which contain only content nodes are ‘openable’. For those containers which have sub-containers, whether they are ‘openable’ or not will be determined on the fly by comparing the granularity level of the content segments they correspond to and the display capability of the terminal device that makes the request. If
the granularity level of a content segment is much coarser than the terminal device’s display capability, the respective container will be set to ‘not openable’ for this terminal device; otherwise it will be ‘openable’. In practice, we can set if the width of a content segment is twice larger than the width of the terminal device’s screen or the height of the content segment is four times larger than the height of the terminal device’s screen then the respective container is ‘not openable’. The ratio we give here is just based on estimation and should be optimized in practice by testing with different types of terminal devices. Thus, the determination of whether a container is ‘openable’ is dynamic while the determination of whether a container is ‘expandable’ is static.

### 3.4 The Layered View of the CCT Data Model

In the forgoing CCT data model, we did not define the data format for the content nodes. Theoretically, the content nodes may contain any type of data no matter they are text or binary. In practice, the data on the web is organized with hypertext documents. As this thesis is targeted at web content adaptation, we will focus our discussion on the application of the CCT data model to the HTML web pages.

Since an HTML web page has its internal structure and can be expressed as a DOM [15] tree, we define a layered view of the CCT data model for a clearer understanding of how the CCT data model work with HTML web pages. That is, instead of viewing the content nodes and container nodes in the same tree, we view them as two different trees with mapping relations between the nodes on the two trees. From the layered view point, we can divide the CCT data model into two sub layers, the data layer and the semantic layer. The data layer contains the HTML DOM tree. The semantic layer
contains the container tree which is similar to the CCT tree described in the previous session except that all the content nodes are removed. Each container node maps to one to many sub-trees in the data layer, and the sub-trees mapped from the parent container node always contain the sub-trees mapped from the children containers. However, the sub-trees mapped from two sibling container nodes should not overlap with each other because the segmentation of a web page is not overlapped. Moreover, overlapping containers will violate the “well formed” rule of the XML syntax and make our embedded implementation impossible. We will discuss the implementation of the CCT data model in the next chapter. This layered structure provides us a three dimensional view of the CCT data model which has two flat tree structures interconnected with each other.

In the layered view of the CCT data model, the container tree in the semantic layer can be expressed as a triple \( \omega = \{\Omega, \Phi, \Delta\} \), in which the definition of \( \Omega \) and \( \Delta \) remains the same to the forgoing CCT data model. \( \Omega = \{\omega_1, \omega_2, \ldots, \omega_n\} \) defines a finite set of containers which can be recursively defined with \( \omega_i = \{\Omega, \Phi, \Delta\} \). \( \Delta = \{\delta_1, \delta_2, \ldots, \delta_M\} \) is a set of rules that defines the relationship among the sub-containers of the current node. The definition of \( \Phi \) is slightly different with that of the CCT data model. In the layered view of CCT data model, \( \Phi \) defines a set of sub-trees of the content DOM tree rather than a set of content nodes. \( \Phi = \{\phi_1, \phi_2, \ldots, \phi_T\} \), in which each \( \phi_i \) defines a sub-tree in the content DOM tree. Every two sub-trees \( \phi_i \) and \( \phi_j \) are mutually exclusive, which means they do not share any common nodes. This is because in a DOM tree, a node can have as most one parent node.
Figure 3-4 shows the layered view of the CCT data model. It consists of two layers, the data layer and the semantic layer. The HTML DOM tree of the web page lies in the data layer; and the container tree lies in the semantic layer. The arrows from the semantic layer to the data layer indicate the mapping relations between the container nodes and their respective DOM sub-tree. Each sub-tree in the data layer starts from the node which is pointed by an arrow from the semantic layer. For example, the root node of the container tree maps to the entire content DOM tree, the root node’s first child container node maps to two sub-trees of the DOM tree in the data layer, and the other child node of the root of the container tree maps to another sub-tree of the DOM tree. Figure 3-4 also indicates that, the sub-trees mapped from the parent container node always consist of the sub-trees mapped from the children container nodes, as we have mentioned earlier.

The layered CCT data model provides an alternative view of the CCT data model. It clearly separates the semantic structure of the document from the document content.
itself, while still maintains the internal structure of the content. The container tree in the semantic layer conveys the semantic and structure information of the web page. The DOM tree in the data layer conveys the content data for end users. By applying a layered view, it is easier to understand how the CCT data model can be used for automatic web content adaptation by processing the information in the semantic layer to locate and retrieve portions of the web content in the data layer.
CHAPTER 4

IMPLEMENTATION OF CCT WITH XHTML

4.1 Different Approaches to Implement the CCT Data Model

We have several choices of the approaches to implement the CCT data model for HTML web pages. First, we could embed a set of special elements in an HTML web page to label the beginning and the end of the segments. Second, we can add some attributes to the existing elements in an HTML web page to indicate whether or not these elements are used for labeling content segments. Finally, we can define a separate reference document to describe the hierarchical segmentation structure of an HTML web page.

Each of these approach options has its pros and cons. The embedded element approach conforms to the layered view of the CCT data model. A container is labeled by the starting and closing tags of a special container element. All the HTML elements between the starting tag and closing tag of a container element will be regarded as the DOM sub-tree this container corresponds to. A container element can be nested into another container element to form the hierarchical structure of the container tree. The advantage of this embedded approach is its simplicity for implementation. The mappings between the container tree and the content DOM tree are implied by the nesting relations of the container elements and the HTML elements. The disadvantage of this approach is that it introduces a new element into HTML language so that further validation may be needed to avoid breaking the standard.
The attribute approach does not require dedicated elements to define the range of
the containers. Instead, it utilizes the existing HTML elements to label the beginning and
ending positions of the containers. To distinguish those HTML elements which are used
for labeling containers, new attributes are needed. In the attribute approach, an HTML
element may serve as a node both in the content DOM tree and in the container tree at the
same time. Thus, the mapping relations are easy to define. The main shortcoming of the
attribute approach is that the definition of a container is constrained by the structure of
the HTML document and sometimes it is even impossible to find a proper HTML
element in the document to serve as the container for a particular segment. For example,
we may have two parallel text paragraphs which are marked as “<p> paragraph 1 text
</p> <p> paragraph 2 text </p>”, and we want to define a container to contain these two
paragraphs; since there is no such an HTML element which is the exact parent node of
only these two paragraph elements, it is impossible to define the container with the
attribute approach. Another concern of this approach is its inefficiency for processing.
Because each HTML element is potentially a container, we have to test all the HTML
elements to find out which of them are containers. Moreover, although the attribute
approach does not require new container elements, it does require new attributes for the
existing HTML elements to indicate whether they are containers. Thus, further validation
is still necessary. Comparing the attribute approach with the embedded approach, the
attribute approach does not expose any advantages over the embedded approach. On the
contrary, it is less flexible and efficient, so we rule this approach out of the options.

The separate reference document approach directly conforms to the layered view
of the CCT data model. In this approach, the original HTML document will be left
untouched, and a separate document will be created to describe the hierarchical structure of the containers and mapping relationship between the container nodes and their respective sub-trees in the HTML document. The reference document can be generated as an XML document with the container elements to form the container tree. A container element can have attributes and children elements to describe its properties and define the HTML sub-tree it maps to. The separate reference document approach does not perform any modifications to the original HTML document. All information in the semantic layer of the CCT data model is kept in the separate reference document, so no validation is needed to keep the HTML standard unbroken. Moreover, because the reference document is dedicated for describing the container tree, it can be designed to be highly efficient for the semantic layer processing without intervention with the data layer. The main problem of this approach lies in the difficulty of defining the mapping relations with the HTML DOM sub-trees. For a complex web page with hundreds of elements, the expression to locate a particular element in the HTML document can be very complex too. For this reason, we keep the separate reference document approach as an option and will study how we can implement the CCT data model with this approach in the future work.

In this thesis, we will focus on the embedded approach due to its simplicity.

4.2 New Container Element in XHTML

Because the HTML syntax is not strict, we apply the Extensible Hyper Text Markup Language (XHTML) [16] for the embedded approach to implement the CCT data model. The advantages of XHTML include:

- XHTML is an official W3C recommendation.
• XHTML is applied by the WAP 2.0 as the specified markup language for its Wireless Application Environment.

• XHTML is a stricter and cleaner version of HTML. Transcoding between HTML and XHTML is easy.

• XHTML strictly conforms to XML syntax, so it can be processed like a XML document.

• XHTML is modular, so extensions can be added to XHTML using XML without breaking the XHTML standard.

Because XHTML is just a reformation of HTML in XML, we can use conversion tools such as TIDY [18] to convert an HTML document into an XHTML document before applying the CCT data model on it. To implement the CCT data model, we introduce a new element named “container” for embedding into XHTML. The container element is used to define the range of a container. The tag <ct> is used to represent the container element. The definition of the container element is given below.

**Definition**

```xml
<ct
  id = "unique alphanumeric identifier"
  label = "index title "
  desc = "content summary text"
  gran = "level of content granularity"
</ct>
```
Element Specific Attributes

- **id (required)**
  This attribute defines a unique alphanumeric identifier for the container element.
  The id attribute is used to locate a container.

- **label (optional)**
  This attribute contains a short text serving as the index of the container. The label attribute should be meaningful and descriptive about the respective content in the container, because it will be used to generate the hierarchical index for the user interface.

- **desc (optional)**
  This attribute contains a summary text of the content in the container. It can be used in fast keyword search or displayed as complementary information to the label index.

- **gran (optional)**
  This attribute defines the granularity level of the content in the container. The granularity level can be defined according to width and height of the respective content segment and the W3C’s Composite Capability/Preference Profiles (CC/PP) [19] or J2ME Profile [20] to provide the information about for what devices the content in this container is suitable. The value of the granularity level is determined during the segmentation process. And this value will be used during the adaptation process to determine whether a container is ‘openable’ for a particular terminal device. The lower the granularity level is, the larger the screen.
size of the terminal device is needed to view the content. For instance, the granularity level ‘1’ means the content segment is only suitable for rendering on the screen of a desktop computer with at least 800 * 600 graphical display resolution; granularity level ‘2’ means the content segment is suitable for PDAs with at least 400 * 300 graphical display resolution; granularity level ‘3’ means the content segment is suitable for smart phones with 200 * 150 graphical display resolution; granularity level ‘4’ means the content segment is suitable for cell phones with 100*75 graphical display resolution. When a granularity level of a container is higher than the designated granularity level for a terminal device, this container is ‘openable’ for that terminal device; otherwise it is ‘not openable’.

Table 4-1 shows an example of granularity definition.

<table>
<thead>
<tr>
<th>Granularity Level</th>
<th>Segment Dimension</th>
<th>Designated Device</th>
<th>‘Open’ eligible devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>800<em>600 to 1600</em>2400</td>
<td>PC</td>
<td>PC</td>
</tr>
<tr>
<td>2</td>
<td>400<em>300 to 800</em>600</td>
<td>PDA</td>
<td>PC, PDA</td>
</tr>
<tr>
<td>3</td>
<td>200<em>150 to 400</em>300</td>
<td>Smart phone</td>
<td>PC, PDA, Smart phone</td>
</tr>
<tr>
<td>4</td>
<td>100<em>75 to 200</em>150</td>
<td>Cell phone</td>
<td>PC, PDA, Smart phone,</td>
</tr>
</tbody>
</table>

Table 4-1: Example granularity definition

Contents

The <ct> tag encloses a piece of XHTML code chunk, which should be well formed according to the XHTML syntax.

Example

A CCT document example is given below in figure 4-1 based on the web page of figure 3-2 and its respective container tree structure shown in figure 3-4. The nested container elements form the container tree. The omitted parts in the example are the HTML
elements from the original web page which form the content nodes of the CCT data model.

```html
<html>
<head>
    ......
</head>
<body>
    <ct id="0" label="header bar" desc="logo and site links" gran="3">
        ......
    </ct>
    <ct id="1" label="left column" desc="site navigations" gran="3">
        ......
    </ct>
    <ct id="2" label="main content" desc="main text and graphical content" gran="2">
        <ct id="3" label="welcome to UBC" desc="photo of the two campus" gran="3">
            ......
        </ct>
        <ct id="4" label="UBC to Establish Okanagan Campus" desc="spotlight text" gran="3">
            ......
        </ct>
        <ct id="5" label="Trek 2010: Green Paper" desc="news callouts" gran="3">
            ......
        </ct>
        <ct id="6" label="More UBC News" desc="news bulletins" gran="3">
            ......
        </ct>
    </ct>
    <ct id="7" label="right column" desc="Quick Find" gran="2">
        <ct id="8" label="Quick Find" desc="hyper links" gran="3">
            ......
        </ct>
        <ct id="9" label="Search UBC" desc="search engines" gran="3">
            ......
        </ct>
        <ct id="10" label="Wayfinding at UBC" desc="searchable campus maps" gran="3">
            ......
        </ct>
    </ct>
    <ct id="11" label="footer bar" desc="more information about contact, copyright" gran="3">
        ......
    </ct>
</body>
</html>
```

Figure 4-1: Container element coding example
4.3 Validation and Implementation Guidelines

Since we introduce a new element to XHTML in the embedded approach, we need to update the XHTML Document Type Definition (DTD) to reflect the change so that an XHTML document with CCT implementation can be correctly validated. To add the new container element into the XHTML DTD, we simply need to add the following lines in the DTD:

```xml
<!ELEMENT ct ANY>
<!ATTLIST ct id ID #REQUIRED>
<!ATTLIST ct label CDATA #IMPLIED>
<!ATTLIST ct desc CDATA #IMPLIED>
<!ATTLIST ct gran CDATA #IMPLIED>
```

In this thesis, we use CCT only for structuring the intermediate document. Thus, we can build our own DTD by extending the XHTML DTD with the new element we introduced and validate the intermediate CCT document against this DTD.

There is no need to update the standard XHTML DTD, because the adaptation results for the end user will not contain the container elements. The `<ct>` tags will be removed from the content segment by the adaptive transcoding module after it is retrieved from the intermediate CCT document.

To implement the CCT data model in an XHTML document with the embedded approach, there are some rules of thumb to follow.

1. The segmentation rule: The segmentation of a web page should be based on both the visual cues and logical cues. Segmentation process should be iterative from the coarsest granularity level to the finest granularity level. Segmentation process
can stop when either there are no more cues available to further divide a segment
or the granularity level of the segment has reached the finest granularity level by
predefinition.

2. The well-formed rule: The positions for embedding the container elements are
usually determined by the segmentation algorithm. However it should be noted
that embedding the container elements should not violate the well-formed rule of
XHTML syntax. The embedded container elements must be properly nested with
other XHTML elements. If the position for embedding a container element
determined by the segmentation algorithm violates the well-formed rule, it should
be adjusted to another position that conforms to the well-formed rule.

3. The labels rule: As the labels of the containers will serve as the index for the user
interface, it is better to be meaningful and descriptive. If the CCT document is
designed by a human, the human designer should choose short and descriptive
text which best tell the nature of the respective content segment. If the CCT
document is generated automatically by a program, it is desirable that a brief
summary of the content can be generated with some artificial intelligence
algorithm to be used as the labels of the containers. However, before an algorithm
which can produce satisfying results is invented, we can use the absolute position
of a segment in its parent segment with its shape as its label, for example “header
stripe”, “left column”, “center box” etc. Finding techniques for effective content
summarization will also be a future work for this thesis.
4.4 User Interface Design

To enable the user to efficiently traverse the embedded tree structure of the containers and retrieve the content segment at desired granularity, we design a multilevel hierarchical index user interface. The end user could take advantage of this multilevel index to navigate a web page and locate the target content segment.

The multilevel index can be generated dynamically as follows: At the beginning, all children containers of the root node (first level containers) are listed, and then wait for the user’s instruction. A container can be “expanded” or “opened” if applicable. If a container is “opened” by the user, its respective content DOM tree will be retrieved, the process is over. If a container is “expanded” by the user, its immediate sub-containers will be listed and then wait for the user’s instruction again. The process iterate until the user choose to open a container. The pseudo code for this process is presented in figure 4-2.

```plaintext
Set current_node = root;
Loop {
    list_children (current_node);
    //wait for user’s instruction
    Set select_node = the container user selected;
    Set select_mode = the operation user selected;
    If (select_mode == “open”) {
        retrieve_content (select_node);
        End loop;
    } else {
        Set current_node = select_node;
    }
}
```

Figure 4-2: CCT container index user interface generation
Figure 4-3 shows the user interface for the example UBC homepage of figure 3-2.

Panel (a) in figure 4-3 shows the main index page that will be returned when the web page is required for the first time. In the index page, the user can open a container by tapping on the respective label or expand a container by tapping on the plus symbol “+” after it when applicable. For those containers which are not ‘expandable’, no plus symbol will be displayed after the label, for instance the “header bar”, “footer bar” and “left column” containers in panel (a). For those containers which are not ‘openable’ for the current terminal device, their index labels will not be underlined. For instance, in panel (a) the “main area” container and the “right column” container are not underlined because the supposed terminal device is a smart phone which is at a higher granularity level than the containers. The sub-container index as shown in panel (b) will be returned when the “main area” container is expanded by tapping the ‘+’ symbol after it. If the “more UBC news” container in panel (b) is opened by tapping on its index label, the content page as shown in panel (c) will be returned.
Additional navigational tools can be provided in the user interface to assist the user to move around. For example, we may provide the following quick links at the bottom of each page.

- Left arrow “←” expands the previous sibling container of the current node if applicable, otherwise open it if applicable.
- Right arrow “→” expands the next sibling container of the current node if applicable, otherwise open it if applicable.
- Up arrow “↑” expands the parent container of the current node.
- Home link, go to the initial index page.

For instance, suppose the current page is the content page as shown in panel (c), the user can display the article in the container “Trek 2010” by tapping the left arrow; the user can go to the index page of panel (b) by tapping the up arrow; the user can go to the main index page of panel (a) by tapping the home link. Since there is no next sibling to the current container, the right arrow will take no effect in this page.

### 4.5 Content Extraction from CCT Documents

With the CCT data model, we can apply XML technologies such as XPATH and XQUERY to easily extract the target content from the respective container. And it is easy to generate the container indexes as described in the previous section (the user interface).

For example, we can create an index of the containers by listing their label attributes. An XPath query like “ct/@label” can return the label attributes of all children containers in the current node.
To generate the index as shown in the figure 4-3 (b) from the CCT document of figure 4-1, we can evaluate the XPath query “//ct[@id = ‘2’]/ct/@label” to get the labels of each children container of the “main content” container.

While the forgoing example can be regarded as “expanding” a container, the XPath query “//ct[@id = ‘container id’]/” can be used to “open” a container and return the respective content segment of that container. For instance, to get the content displayed in figure 4-3 (c) from the CCT document of figure 4-1, the XPath query “//ct[@id = ‘6’]/” should be evaluated.

We can use XSLT to format the XPATH query results and generate web pages suitable for displaying on the terminal devices. For example, the XSLT script as shown in figure 4-4 can generate a container index page as shown in figure 4-5.

```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="/">
 <html>
  <head>
   <xsl:copy-of select="//head/*"/>
  </head>
  <body>
   <h2><xsl:value-of select="//head/title"/></h2>
   <ul>
    <xsl:for-each select="/body/ct">
     <li><xsl:value-of select="@label"/></li>
    </xsl:for-each>
   </ul>
  </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Figure 4-4: An XSLT script example for generating container index page
Similarly, we can use XSLT to generate and format a content page. Since the HTML elements for visual formatting are kept in the content nodes, the resulting content page will preserve the visual appearance of the original page if the browser supports HTML. Otherwise, transcoding may be needed to convert the content page into the coding scheme that is supported by the browser. While XHTML is the standard markup language in WAP 2.0, we assume it is supported by most types of the terminal device. The XSLT script as shown in figure 4-6 will extract the content of a sub container in the right column container, and the result in the content page as shown in figure 4-7.

```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="/">
 <html>
  <head>
   <xsl:copy-of select="/head/*"/>
  </head>
  <body>
   <xsl:copy-of select="/ct[@id = '9']="/>
  </body>
 </html>
</xsl:template>
```

Figure 4-6: XSLT script for extracting content node
Note that we use the `<xsl:copy-of>` element instead of the `<xsl:value-of>` element in the example XSLT script to preserve the HTML elements of the original web page.

4.6 Extraction of the Embedded Container Tree

In the embedded CCT implementation with XHTML, the `<ct>` tags are mingled with other HTML tags. So, there may be zero to many elements between two container elements. For example, we may want to make an HTML table the content node of the `<ct id = “table”>` container element, and we also want to put the content of each row of the table in a `<ct id = “row number”>` container element. To keep the completeness of the table structure, the table element should be the child node of the `<ct id = “table”>` container element; to conform to the well-formed rule, the `<ct id = “row id”>` elements should be placed in the middle of the starting tag `<tr>` and the closing tag `</tr>` for the table row elements. So the resulting CCT document will look like the example code in figure 4-8.
In the above example, the <ct id = “row number”> container elements are descendents of the <ct id = “table”>. They are children of the respective table row elements, which are children of the table element, which in turn are child of <ct id = “table”> container element. But from the layered view point of the CCT data model, we do not care how many table row elements or table elements or any other HTML elements are in the middle of the hierarchy of two container elements; we view all the container elements in a separate tree in the semantic layer. Thus, those <ct id = “row number”> container elements are immediate children of the <ct id = “table id”> container element in the container tree, and they are sibling nodes themselves.

When we use an XPath query such as“.//ct” to locate a <ct> element, we should clarify that the query is evaluated to the container tree only. Because there may be other elements between the current node and the target <ct> element in a DOM tree, we should use XPath query like “.//ct[with no other container nodes between the current node and the target node]” for the DOM tree to express the same query of “.//ct” for the container tree. Unfortunately, such query cannot be expressed in XPath 1.0 because it needs universal quantifiers to express which are not supported by XPath 1.0. The good news is, the universal quantifier will be supported in the coming XPath 2.0.

Then, there are basically two methods to query a container element. The first method is to extract the container tree from the embedded CCT document first with text processing tools, and then apply XPath query on the container tree to locate a container.
Once a container is located, we can use its container id to extract its content nodes from the embedded CCT document. The second method is to apply XPath 2.0 query with universal quantifiers on the embedded CCT document directly, ignoring any element which is not a container. Each of the two methods has its pro and con. The first method is more efficient for querying a container node because the separate container tree is much more compact than the embedded CCT tree. The second method is less efficient for querying a container node due to the large scale of the embedded CCT tree, but it does not require pre-processing to extract the container tree. Which method to apply will much depends on the specific architecture of the content adaptation system.
CHAPTER 5

CCT APPLICATION DESIGN AND EXPERIMENTS

5.1 CCT Adaptive Web Server Design

According to the CCT content adaptation framework, we developed an adaptive web server to test the feasibility and the effectiveness of the CCT data model. This CCT adaptive web server allows the user to navigate through the container indexes of a CCT document and display the target content segments at a desired granularity level. We implement the application as a WBI plugin. WBI is a programmable web proxy developed by IBM that enables developers to create intermediaries to tailor, customize, personalize, or enhance data as they flow along the HTTP stream. For detail introduction of WBI, please refer to the section 2.2.7 of this thesis.

The adaptive web server application consists of two MEGs (The Monitor, Editor, Request Editor and Generator modules are collectively referred to as MEGs in WBI). One is the ServerFileGenerator and the other is the CctXslProcEditor. A plugin named CctWebServerPlugin is used to hold and initialize these two MEGs. The ServerFileGenerator corresponds to the web server in the CCT content adaptation framework. The CctXslProcEditor corresponds to the adaptive transcoding module in the framework. The function of the proxy module in the framework is carried out by a default proxy MEG provided by WBI.
The system works in this way. When a client sends a HTTP request for a file to the CCT web server, an object of ServerFileGenerator will be instantiated. The ServerFileGenerator object will generate a HTTP response header and put the requested file into its output stream. The HTTP response header and file content will be sent to the client directly unless certain rules are matched to trigger the CctXslProcEditor. In this application, we set the rule to be “the requested content is of CCT document type.” If this rule is matched which means a CCT document is returned by the generator, an object of the CctXslProcEditor will be instantiated. This editor object will load a template of XSLT style sheet into the memory, and modify the style sheet based on the parameters retrieved by the generator from the original HTTP request header and passed through with WBI transaction data mechanism. Then the modified style sheet will be applied on the requested document coming from the MegInputStream to generate proper result content which flows out through the MegOutputStream. The output of the CctXslProcEditor is either a container index page or a content page depending on what kind of query to evaluate to the CCT document.

To retrieve a document from the local file system, we take use of a WBI MEG bean named FileGenerator provided by the WBI API. The ServerFileGenerator simply analyzes the input HTTP request header, generates file path based on the request and forward the file path and the request to the FileGenerator bean. The FileGenerator will take care of getting the document from the local file system and put the content into the output stream. Figure 5-1 shows the UML diagram of the ServerFileGenerator.
The `CctXslProcEditor` is responsible for performing the XSLT transformation on the CCT document to generate adaptive content. In our design, we use DOM and JAXP for realizing such functions. The `CctXslProcEditor` applies the DOM API in the org.w3c.dom package for online modification of the XSLT style sheet template and the JAXP API in the javax.xml.transform package for XSLT transformation on the input CCT document. Figure 5-2 shows the UML diagram of the `CctXslProcEditor`. 
The XSLT style sheet template contains two sub templates, template ‘A’ for generating the index pages, and template ‘B’ for generating the content pages. For easy online modification, each template defines two parameters, the ‘source’ and the ‘xpath’.
The source refers to the CCT file name along with its relative path under the web server root, which is needed to create href values of links. The xpath refers to the XPath expression for the interested container node, which is needed to extract proper content from the source file. The value of these two parameters will be modified by the CctXslProcEditor on the fly to set to the run time values. The default values in the stylesheet are for initial request with no queries. The XSLT style sheet source code is listed below.

stylesheet.xsl

```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="/">
    <html xmlns="http://www.w3.org/1999/xhtml">
      <head>
        <xsl:copy-of select="/head/ct/*"/>
      </head>
      <body>
        <xsl:call-template name="A">
          <xsl:with-param name="source">url of the source file</xsl:with-param>
          <xsl:with-param name="xpath" select="/body"/>
        </xsl:call-template>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```
To correctly set the parameters for the XSLT style sheet, it is necessary to get the information such as the interested container id and what kind of operation on it. This information can be passed with the HTTP GET method. The GET method can pass key values pairs in UML requests. For example, the key is either “container” or “content”, and the value is an identification expression for the container node of interest. When the key is “container”, it means an index page of the children containers is requested; when the key is “content”, it means a content page of current container is requested. The CCT identification expression is in the format “ct[@id = #]”. The reason we use the round bracket instead of the bracket in an XPath expression is because the bracket will be misinterpreted by the Nokia WAP gateway simulator for other function. An example of such an HTTP request is like:

5.2 Extending the Adaptive Web Server for Automatic Content Adaptation

Comparing the CCT web server application architecture with the proposed CCT content adaptation framework, the ServerFileGenerator corresponds to the web server in the framework. The CctXslProcEditor corresponds to the adaptive transcoding module in the framework, although it lacks of the markup language transcoding or image conversion functions. These functions can be easily accomplished by adding new editors in the system to further manipulate the output of the CctXslProcEditor. This is one of the advantages of applying the WBI. The function of the proxy module in the framework is carried out by a default proxy MEG provided by WBI. Since the CCT documents are manually generated rather than automatically transformed from a plain HTML document, there is no respective MEG in this adaptive web server application corresponding to the content reformation module of the CCT content adaptation framework. However, this function can be realized by adding new editors that can automatically transform an HTML web page into a CCT document with vision-based page segmentation algorithms to achieve dynamic automatic content adaptation.

5.3 Test Bed Setup

To demonstrate the effectiveness of the proposed CCT data model, we setup a test bed for accessing the adaptive web server from heterogeneous terminal devices. The test bed applies the server-client architecture. The server is a PC running the adaptive web server application. The client can be either the Nokia mobile browser simulator or the
Microsoft’s IE web browser running on another PC on the same Local Area Network (LAN) with the server. The IE web browser is a typical web browsing software on PC. According to the definition in table 4-1, we set the designated granularity for the IE client to ‘1’. The Nokia mobile browser simulator simulates the work of the web browser on a smart mobile phone. We set the designated granularity for the mobile browser simulator client to ‘3’. To enable the communication between the web server and the mobile browser simulator, a Nokia WAP gateway simulator is also needed on the same machine of the mobile browser simulator. Figure 5-4 shows the configuration of the test bed.

Figure 5-4: Test bed configuration

Software used in the test bed includes:

- WINDOWS 2000
- WBI 4.5
5.4 Experiment Results

We created two CCT documents for test. The first one is the UBC homepage as shown in figure 3-2. The second one, which is more complex, is the YAHOO homepage. Some necessary modifications to the original HTML pages are made to comply with the CCT data model. First, as CCT documents are essentially XHTML documents, we cleaned the original HTML file with TIDY [18] to make it well-formed. Then, we embedded the <ct> tags into the proper places in the documents to reveal the hierarchical structure of the content. Finally, we changed the file type of the document to CCT.

5.4.1 Experiment with UBC Homepage

Figure 5-5 shows the result of browsing the UBC homepage from both the Nokia mobile browser simulator and the IE browser. Panel (a) in figure 5-5 shows the main index displayed on the mobile browser when the web page is requested for the first time. The main index page displayed on the IE browser is shown on panel (b). Comparing the two index pages on different browsers, we can notice that the “main content” label and the “right column” label are not hyperlinked in the page for the mobile browser, which means they are not ‘openable’ for this terminal device. This is because the granularity
levels of the “main content” container and the “right column” container are both ‘2’, but the designated granularity level for smart phone is ‘3’ which is higher than the granularity levels of the two containers. Thus, according to our definition on granularity levels in table 4-1, the “main content” container and the “right column” container are not ‘openable’ for this smart phone mobile browser. On the other hand, because the designated granularity level for PC is ‘1’, which is lower than the two containers’ granularity level, so they are ‘openable’ and hyperlinked in the page for the IE browser.

There is one more “full page” link in the main index page for IE which is not presented in the mobile browser. This link corresponds to the root container which contains the whole web page and has granularity ‘1’. Since the designated granularity level of PC is also ‘1’, the whole page can be displayed by the IE browser on PC. The granularity level for the other three containers “header bar”, “left column” and “footer bar” are all ‘3’, so they are ‘openable’ for both the IE browser and mobile browser.

Each label in the index represents a CCT container. Some labels are followed by a plus symbol ‘+’, which means that container is ‘expandable’. Tapping on the plus symbol will expand the container and generate an index page of its sub-containers. For example, panel (c) of figure 5-5 will be displayed on the mobile browser if the plus symbol after the “Main Content” label is tapped. Panel (d) shows the index page of the sub-containers of the “Main Content” container on the IE browser. Because the sub-containers of the “Main Content” container are all at granularity level ‘3’, they are all ‘openable’ on both the mobile browser and the IE browser. Tapping on a hyperlinked label will retrieve the container’s respective content segment. For example, the content segment in panel (e) will be displayed on the mobile browser by tapping on the “more UBC news” hyperlink.
in the index page of panel (c). Panel (f) shows the same content page as panel (e) but in the IE browser.
One significant advantage of the CCT adaptive web server is that it gives users the freedom to choose the content segment at any granularity levels as long as it is supported by their terminal devices which means the granularity level of the content segment is equal to or higher than the designated granularity level of the terminal device. For the users with large screen devices, it is not necessary to go through all the levels of sub-indexes to get the desired content, the user can choose to tap on the label hyperlink rather than the plus symbol at a proper level to get the whole content segment at once. For example, the user with a PDA with granularity level ‘2’ can tap the “Right Column” label rather than the following plus symbol to get all the contents in the right column in one page as shown in panel (a) of figure 5-6.
Figure 5-6: Granularity hierarchy of the content in the right column container.

Figure 5-6 panel (b) shows the container index of the right column, and figure 5-6 panel (c), panel (d) and panel (e) show the respective content page of the containers in the container index page of panel (b). In some cases, although the user has a large screen device, he may also want to follow the multilevel indexes to go to the content segment at
higher granularity level than his terminal device to quickly locate the target information. This is pretty useful when the connection speed is slow, for instance using a dial up connection with notebook computer or PDA, because large scale of irrelevant contents can be filtered by following the indexes to get to the target content directly.

5.4.2 Experiment with Yahoo Homepage

Many of today’s web pages rely on the `<table>` element to organize their layouts. The homepage of Yahoo is a typical example of extensively using tables for visual organization. Such usage of tables introduces complexity into the web pages; however we’ll prove by our example of the adaptive Yahoo homepage that the CCT data model can handle such complexity successfully. In the example, all the page layout structural information of the original page is kept. Thus, ‘almost’ the same web page of the original one can be generated from the CCT document. We say ‘almost’ because some page information such as the HTML header information may be modified to suit the needs of the terminal devices, but the main body of the page will remain the same. In this example, the “full page” link in the main index page corresponds to the root container and allows the user to view the whole page from a PC or any equivalent powerful device. The drawback of keeping all the layout structures such as tables, space and background images is that it may cause troubles for some less powerful device to render the content correctly. However, this can be solved by adding more transcoding entities to further filter or modify such elements from the result according to the terminal device’s profile. Figure 5-7 shows the test results of accessing the Yahoo homepage on the CCT adaptive web server from the IE web browser.
Figure 5-7: The Yahoo homepage on the CCT adaptive web server

Figure 5-7 panel (a) shows the main container index of the adaptive Yahoo homepage. The full page in figure 5-7 panel (b) will be displayed by tapping the “Full page” link in the main index page. Figure 5-7 panel (c) shows the sub-container index of
the right column; and the content of the “Market place” container is displayed in figure 5-7 panel (d).

5.4.3 Conclusions on the Experiment Results

These experiments prove the feasibility and effectiveness of the proposed CCT data model and the CCT content adaptation framework. These experiments also demonstrate the following advantages of applying the CCT data model for web content adaptation.

1. Easy content generation: A CCT document complies with the XHTML syntax so it is easy to create just like normal XHTML documents;
2. Easy application implementation: The CCT application applies the classical web architecture. No changes are needed for the client and server architecture. Adaptive results are generated on the fly;
3. User awareness: The user is involved in the adaptation process for deciding the target content segments and the granularity level, only useful data to the end user will be delivered. The hierarchical index makes it easy for the user to navigate through the contents. The flexible opening and expanding methods allows the user to adjust the granularity;
4. Speed: Due to the smaller size of each sub page, it is less error-prone during transmission and faster to load for browser.
CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this thesis, we have proposed a novel data model named Content Container Tree (CCT) for the web content adaptation. The CCT data model organizes the semantics and the content data with an embedded tree structure. By “embedded tree”, we mean both the containers and contents can be organized as independent tree structures; and the container tree can be embedded into the content tree to indicate the mapping relations between the containers and the content segments. The container tree can be processed independently to generate multilevel indexes for navigating through the web content segments at various granularity levels. A content segment at the proper granularity level can be retrieved according the terminal device’s characteristics and the user’s input.

We have also defined a layered view for the CCT data model, which consists of a semantic layer and a data layer. The semantic layer carries the semantic and logical information about the web content with the container tree, while the data layer carries the content data with the content tree. The layered view of the data model enables the architectural separation of the high level adaptation and the low level adaptation for the web content adaptation framework. We have defined the high level adaptation as the adaptations on structure, navigation, and functions of the web contents which should be processed in the semantic layer. We have defined the low level adaptation as the adaptations on the syntax, format and encoding of the content which should be processed in the data layer.
The explicit separation of the high level and low level adaptation results in a flexible and extensible modular web content adaptation framework which we also introduced in this thesis. This framework conducts the high level adaptation and the low level adaptation with different modules and organizes the intermediate adaptation result with the CCT data model.

In conclusion, the major differences between the CCT approach and other approaches to web content adaptation include:

1. The CCT approach is data oriented while other existing approaches are process oriented. Existing web content adaptation approaches mostly focus on finding and applying particular techniques to transform a web page based on the terminal device characteristics. The CCT approach, on the other hand, focuses on defining a data model that carries the necessary information and the data for adaptively generating web pages according to the user’s preference and the terminal device’s characteristics. By using the CCT as the intermediate data structure, the content adaptation process can be divided into two steps. The first step is to transform the original web page to conform to the CCT data model. The second step is to retrieve the content segments at the desired granularity levels from the intermediate CCT document and perform further transcoding if necessary.

2. The CCT approach supports multilevel granularity of the web contents, while other approaches only has one level of content granularity. The existing approaches define only one granularity level of the content according to the terminal device’s screen size. For instance, in the Microsoft’s thumbnail view browsing system, the dimension of the sub-pages is pre-determined by the screen
size of the supported PDA. But these sub-pages may not be suitable for viewing on other devices with larger or smaller screens. The CCT approach, on the other hand, supports multi-level granularity. The highest granularity level is determined by the smallest screen size supported by the system. The hierarchy of the segments at different granularity can be utilized to generate multilevel indexes to enable the user to navigate around. The user with small screen device can choose the segments at higher granularity level to display while the user with large screen device can choose the segments at lower granularity level to display. The user with large screen device also has the freedom to follow the hierarchical index to go to the higher granularity level to quickly locate the target content and save downloading time.

3. CCT approach is dynamic while other approaches are static. Existing web content adaptation approaches usually pre-generate a series of sub-pages from the original web page and wait for the user to choose from these sub-pages. For instance, MIT’s digestor system re-authors web documents through a series of transformations according to the type of the terminal device, and links the resulting individual sub-pages. Because the user will not access all of these sub-pages in most cases, the processing power is wasted to generate those sub-pages that never accessed. On the contrary, with the CCT approach, a result page will be generated on the fly only when it is requested. After transforming the original web page into the intermediate CCT document, CCT adaptation system will generate an index page to deliver to the terminal device and wait for the instruction from the user. If the user requests to expand a container, the CCT system will query the
CCT document to generate an index page of the sub-containers of the concerned container. If the user requests to open a container, the CCT system will retrieve the respective content segment from the intermediate CCT document to generate a content page. Because the result page is generated on the fly only when it is requested, no processing power is wasted to generate sub-pages that are never accessed, so CCT approach is more efficient.

6.2 Future Work

Our work in this thesis was mainly focused on the design of the CCT data model and the CCT adaptation framework. To achieve automatic web content adaptation, we also need to invent algorithms that can automatically analyze a web page and transform it to the CCT document. There are two main challenges for this work. First, the algorithm should be able to perform iterative web page segmentation based on visual cues as well as logical cues. Second, the algorithm should be able to make summarizations of the content segments to generate meaningful and descriptive labels for the containers. There have been quite a few related works [11-14] on the first challenge that we can refer to. The second challenge is even harder and need more in-depth study. Text summarization techniques as described in [4] can be applied on text-intensive small content segments. But for a large segment with several sub-segments and multimedia content, new techniques should be developed to get the summarization that best describe the content in that segment.

In this thesis, we took the embedded tree approach to implement the CCT data model. In the future work, we should also try other approaches for CCT implementation,
for instance the separate reference document approach we mentioned in the section 4.1. We should investigate the pros and cons of each approach in detail and find out the most optimum approach for CCT implementation.

In this thesis, we only consider the static HTML documents for web content adaptation. However, more and more web pages today are generated dynamically with scripting languages such as ASP (Active Server Page) and PHP (Personal Home Page). Moreover, embedded scripting language such VBScript and JavaScript are also widely used to control the appearance and behavior of a web page. How to handle these dynamic web pages in web content adaptation with CCT is also an important topic for the future work.

Other possible future works of this thesis include developing authoring tools to assist manually creation of CCT documents; comparing the effectiveness and efficiency of the CCT adaptation framework with other web content adaptation approaches; and analyzing the performance of the CCT content adaptation system in the terms of speed, usability and content fidelity.


[18] Dave Raggett, *HTML TIDY*, http://www.w3.org/People/Raggett/tidy/


