Chinese Information Access Through Internet On X-Open System

By

Yao Jian

Supervised By :

Prof. Lu Chin

Submitted to the Division of Department of Computer Science & Engineering

in partial fulfillment of the requirements for the

degree of Master of Philosophy

at the

Chinese University of Hong Kong

June 1997

Chinese Information Access Through Internet On X-Open System

submitted by

Yao Jian

for the degree of Master of Philosophy

at the Chinese University of Hong Kong

Abstract

With the advent of World Wide Web, not only English documents are exchanged in the Internet, the exchange and manipulation of none English documents, including Chinese documents are also in great demand. The problem of Chinese information access is that the large set of Chinese characters and the co-existence of multiple codesets and the incompatibility among them bring out the obstacle to share and exchange of Chinese information for people from Mainland China, Hong Kong and Taiwan.

To provide a friendly environment for people to fetch, exchange and manipulate Chinese documents with codeset transparency, codeset announcement mechanism, automatic codeset conversion are introduced into the current web system. Also the browser provides a Chinese interface for the ease of operating the system.

In this thesis, I describe the design of our Chinese web server and an internationalized browser on X-Open system as well as the implementation results. The web server is built on a Unix platform similar to other English Internet servers with the capability to manage Chinese text data encoded in different codesets on the same server. The server also provides automatic codeset conversion transparently to client machines when data stored in the server are incompatible with what the client machine can process. The internationalized browser allows users to access the server information in its local familiar environment, either using traditional Chinese or simplified Chinese with-

out the need to match the codeset of the documents being retrieved from the server. The browser is built in an internationalized way so that user interface can switch to different language environment easily. The browser also has the capability to handle on-line automatic codeset conversion if necessary. A proxy server with automatic codeset conversion and cache mechanism is also provided for users to communicate with those servers which do not offer Chinese specific services.

To provide friendly and convenient Internet access environment for users who do not have our enhanced web system at hands, a new approach using Common Gateway Interface (CGI) is also devised to realize the codeset announcement and automatic codeset conversion such that users can select the codeset they prefer through the interface web page and the converted document encoded in the codeset preferred by users will be sent back after the CGI program conducts the proper handling.

Acknowledgments

First of all, I would like to give my deepest gratitude to my advisor, Prof. Lu Chin, for her tremendous support, encouragement and guidance throughout my graduate study. She is enlightening, critical and helpful on research, always being ready to discuss problems and results. Many discussions with her helped me to progress in the right direction. She has tirelessly corrected my writing draft after draft, including papers and this thesis. She is always warm and kind to give her hands when I have problems of life during the past two years. I am also very grateful to Prof. Lee Kin Hong who gave me many constructive advices on my research work. His rigorous working style when correcting my research papers impressed me deeply. He is a kind person and gives me a lot of help during my studies here. I greatly appreciate Prof. Kan Wing-Kay, for serving on my defense committee and giving me important comments on my thesis. I would like to thank Prof. Sun Yu Fang, from Chinese Academy of Sciences. I learned a lot of knowledge from text books on C programming language and Unix systems written by him during my undergraduate studies. It is my honour to have Prof. Sun to serve as my defense committee member and I highly appreciate his valuable knowledge and comments on my research work.

I would like to thank all my classmates, past and present, who have helped me when I encountered problems on terminals or softwares. Without their help, it is impossible for me to rapidly grasp so many knowledge and advanced techniques on Unix and X-Window System. I especially appreciate the collaboration with my research group members: Wong Man-fai, Lawrence Nui Pui Tak, Eddie Kwan Hoi-ching, Paul Pang Chi-wang, Johnny Yip Hoi-man, Michael Ng Mau Kit who give me great help, support and advice on many technical issues during the project goes on.

Special thanks to Mr. Lau Sau Ming who gave me many precious advices on my research project and shared all his knowledge, experience in research and even some tricks on programming. He is a warm-hearted and patient friend who is always ready to give me encouragement when I was in low spirit. I am very much indebted to Mr. Lu Si Fei for his help on many professional issues. He discussed with me on my

research project for many times and shared his precious knowledge on world wide web, browser/server computing model and networking. I am also very grateful to Mr. Liu Jian Zhuang for helping me in my thesis editing and working out many figures with me. I greatly appreciate Mr. Pan Jiao Feng for giving me valuable suggestions and pertinent comments to the drafts of my paper and thesis.

Thanks also to many wonderful friends who made my stay in CUHK so pleasant and unforgettable. My special thanks go to Li Hai Ying, Zhang Xue Jie, Li Yuan Yuan, Li Guan Xin, Tian Ying Li, Zhu Zhe Ying, Wang Li Di, Zhang Jian Xin, Wang Zhi Jun, Chen Jian Wei and Wang Wai Ting. I will treasure their friendship for the rest of my life.

I am forever indebted to my families for their unending love and support. My parents are always there when I need them and their constant encouragement makes me confident in what I do. My loving younger brother is always ready to give me help. Last, but not least, I am deeply indebted to my dear husband; without his understanding, sacrifices and patience, the completion of this thesis would not have been possible.

Contents

Abstract i

Acknowledgments iii

1 Introduction 1

2 Basic Concepts And Related Work 6

2.1 Codeset and Codeset Conversion 7

2.2 HTML Language 10

2.3 HTTP Protocol 13

2.4 I18N And L10N 18

2.5 Proxy Server 19

2.6 Related Work 20

3 Design Principles And System Architecture 23

3.1 Use of Existing Web System 23

3.1.1 Protocol 23

3.1.2 Avoid Duplication of Documents for Different Codesets 25

3.1.3 Support On-line Codeset Conversion Facility 27

3.1.4 Provide Internationalized Interface of Web Browser 28

3.2 Our Approach 29

3.2.1 Enhancing the Existing Browsers and Servers 30

3.2.2 Incorporating Proxies in Our Scheme 32

3.2.3 Automatic Codeset Conversion 34

3.3 Overall System Architecture 38

3.3.1 Architecture of Our Web System 38

3.3.2 Flexibility of Our Design 40

3.3.3 Which side do the codeset conversion? 42

3.3.4 Caching 42

4 Design Details of An Enhanced Server **44**

4.1 Architecture of The Enhanced Server 44

4.2 Procedure on Processing Client's Request 45

4.3 Modifications of The Enhanced Server 48

4.3.1 Interpretation of Client's Codeset Announcement 48

4.3.2 Codeset Identification of Web Documents on the Server 49

4.3.3 Codeset Notification to the Web Client 52

4.3.4 Codeset Conversion 54

4.4 Experiment Results 54

5 Design Details of An Enhanced Browser **58**

5.1 Architecture of The Enhanced Browser 58
5.2 Procedure on Processing Users' Requests 61
5.3 Event Management and Handling 63
5.3.1 Basic Control Flow of the Browser 63
5.3.2 Event Handlers 64
5.4 Internationalization of Browser Interface 75
5.4.1 Locale 76
5.4.2 Resource File 77
5.4.3 Message Catalog System 79
5.5 Experiment Result 85

6 Another Scheme - CGI **89**

6.1 Form and CGI 90
6.2 CGI Control Flow 96
6.3 Automatic Codeset Detection 96
6.3.1 Analysis of code range for GB and Big5 98
6.3.2 Control Flow of Automatic Codeset Detection 99
6.4 Experiment Results 101

7 Conclusions and Future Work **104**

7.1 Current Status 105
7.2 System Efficiency 106
7.3 Future Work 107

Bibliography **109**

A Programmer's Guide **113**

A.1 Data Structure 113

A.2 Calling Sequence of Functions 114

A.3 Modification of Souce Code 116

A.4 Modification of Resources 133

B User Manual **135**

List of Tables

2.1 Codesets of Chinese 8

2.2 Code Range of Hanzi Characters in Different Codesets 9

2.3 Language encodings as supported by Netscape 3.0. 20

2.4 Access result using Netscape and CMosaic. 21

3.1 Implementation Possibilities. 33

3.2 Different Codeset Converters 35

3.3 Exceptional Cases in Codeset Conversions 37

5.1 Locales set up in our web system 78

5.2 File Types Related To Message Catalog System 83

5.3 Elements of an NLSPATH Value 85

6.1 Some Useful Environment Variables 94

6.2 Valid HTTP Headers for CGI programs 95

List of Figures

2.1 Client-Server Model used in WWW 7

2.2 Codeset Conversions 9

2.3 The Web Page of a Simple HTML Document 11

2.4 The Web Page of a Simple Form 12

2.5 Negotiation of Data types between Web Browsers and Web Servers . . . 13

2.6 Proxy server act as a client and a server 19

3.1 File Storage Structure on Multi-codesets Web Servers 25

3.2 Output for viewing a Chinese GB web page using a Big5-based browser 27

3.3 Multi-Codeset Conversions of Both Web Browsers And Web Servers. . . 31

3.4 The Architecture of API for Codeset Conversions 34

3.5 The Structure of the Codeset Conversion API 35

3.6 Calling Sequence of the Conversion Routines 36

3.7 Overall System Architecture. 38

3.8 Case I and Case II 40

3.9 Case III and Case IV 41

4.1 The Architecture of The Enhanced Web Server 45

4.2 HTTP/1.1 Request Message From Our Enhanced Browser 49

4.3 Steps in Determining the Codeset of a Given File 51

4.4 Demonstration on <LANG> Tag Identification By The Enhanced Server 55

4.5 Demonstration on Codeset Identification By File Extension 56

4.6 Demonstration on Handling Multi-Codeset Chinese Document 57

5.1 Architecture of The Enhanced Browser 59

5.2 Basic Control Flow of Mosaic. 63

5.3 Control Flow of Event "Open URL." 66

5.4 FSM MIME Parser. 69

5.5 Codeset Conversion Handlement In MIME Parser. 73

5.6 Relationship between language and locale 78

5.7 Relationship Between Locale and Resource File 79

5.8 The Fragment of The Message Source File 82

5.9 Message Catalog Directory Structure In Our System 83

5.10 The Chinese Interface of Our Browser 86

5.11 On-line Codeset Conversion Done By The Enhanced Browser 87

6.1 The Operation Procedure of CGI. 90

6.2 The Interactive Interface Designed For Codeset Announcement 92

6.3 Control Flow of CGI Program. 97

6.4 Big5 and GB encoding table. 98

6.5 Control Flow of Automatic Codeset Detection. 100

6.6 Codeset Conversion From Big5 To GB 102

6.7 Codeset Conversion From GB To Big5 103

A.1 History List 114

Chapter 1

Introduction

In recent years, Internet access has become a common practice. Especially, the emergence of the World Wide Web (WWW), gives us a chance to navigate the world of global information with the click of a button. Meanwhile, the availability of softwares such as Gopher, Mosaic, Netscape and Bulletin Board Systems(BBS) facilitates the dramatic growth of WWW. Academics and business communities alike use Internet to exchange and share information as well as software. However, most of the Internet softwares are designed for English or other alphabet based languages only. Exchange and sharing of Chinese information is very limited although exchange and manipulation of Chinese text information via WWW are in great demand in Mainland China, Taiwan, Hong Kong and other places where Chinese characters are used. With more and more non-alphabet based language web documents available on the Internet such as Chinese documents, specific supports are needed to handle Chinese and other non-alphabet based languages.

As an ideographic language, Chinese requires a much more complex processing environment. The complexity comes mainly from the large set of Chinese characters that a computer system has to process and the fact that there exists more than one computer coded character set, referred to as *codeset*. Different codesets are incompatible because one code value can represent different Chinese characters in different codesets. The GB codeset, for example, which represents simplified Chinese writing, is used in

Mainland China on all computer platforms. However, Big5 and CNS, which represent traditional Chinese writing, are used in Taiwan and Hong Kong for PCs and workstations, respectively. The co-existence of multiple codesets for Chinese characters is not only a nightmare for Chinese software developers as they have to maintain many different versions of the same software, but also a great barrier for people from Hong Kong, Taiwan and Mainland China to communicate with each other over the Internet.

In general case, source Chinese documents on the web are written in different codesets. They are maintained on web servers at different places. Most of the existing web servers which provide Chinese documents store multiple versions for the same Chinese document, i.e. one version per codeset. Each time, users choose to read a certain version through selecting the corresponding codeset from the web page. This method realizes the support of multiple Chinese codesets at the server side at the cost of the large duplication of Chinese documents which wastes a lot of disk space. On the other hand, different web browsers have different capabilities on supporting Chinese codesets. One issue is that although some web servers provide multiple versions for different codesets of Chinese documents, it still depends upon the local support of the web browser whether users can read those Chinese documents or not. Suppose that a web browser on PC platform supports only traditional Chinese codeset - BIG5, then users can access only the BIG5 version of that document. If users try to access Chinese text data from a web server which provides only simplified Chinese documents, there is no way for users to read them with their local browser. In such case, codeset conversion is needed to be carried out either at the server side or at the browser side to convert the document encoded in GB 2312 into the target codeset BIG5 version.

Another special issue for Chinese text processing is that users from different regions may have their own reading preferences. For example, people from Mainland China prefer to read simplified Chinese while people from Taiwan and Hong Kong may like to read traditional Chinese. Although some web browsers support multiple Chinese codesets environment, codeset conversion is also needed to satisfy their reading preferences if the codeset of the original document is incompatible with what users prefer. Furthermore, as a Chinese Internet access tool, the browser interface such as buttons

and menus must be customized in a way that is convenient for Chinese users.

In order to handle Chinese text retrieval via WWW with automatic recognition of codesets and the conversion among them, some *codeset announcement* mechanisms must be provided. We know that Internet access uses the client-server model to manage client access and server information handling separately. WWW, as a popular way of Internet access, uses the client-server model to carry out communication between web servers and web clients (browsers). The protocol used between web clients and web servers is HTTP (HyperText Transfer Protocol) which handles only one request at a time. To carry out codeset announcement, the client side must be able to announce its local environment or codesets supported, the server side must announce the codeset information for documents it manages. If the two announcements do not match each other, automatic codeset conversion can then be supported either before the document is transferred or after it is received on the client side.

The exchange of codeset announcement information between web servers and web clients (browsers), referred to as *data type negotiation*, can be realized through HTTP/1.1 protocol. However, most of the current web servers and web clients communicate with each other via pre-HTTP/1.1, such as HTTP/1.0 [21] protocol, which does not support data type negotiation between servers and clients. It works fine for alphabet based languages since the default codeset in the case of HTTP protocol is ISO-8859-1 (the so-called "*Latin-1*" for Western European characters) [1]. However, for Chinese text data, which is composed of multi-byte characters, the lack of data type negotiation may cause misinterpretation of data. For example, both CMosaic and Netscape 3.0 can access and display Chinese text. However, the Chinese text can be displayed properly only if the local environment of the client, mostly the codeset it supports, is compatible with the codeset of the documents retrieved from the server. There is no automatic way to detect if the local environment of the client machine is compatible with the data retrieved from the server. Users have to manually try out different encodings supported in the browser to interpret the retrieved documents correctly. With the help of codeset announcement and automatic codeset conversion, a web server stores only a single version for each Chinese document, if the local environment of a web browser is

incompatible with the codeset supported by the server, automatic codeset conversion can be done to convert the document from the original codeset to the target one. At last, the converted document is displayed by the web browser properly. This approach provides a more convenient and user friendly environment for users to access Chinese information through Internet and it avoids the duplication of Chinese documents.

In this thesis, I shall describe the design of a Chinese web server and an internationalized browser on X-Open system as well as some implementation details. The Chinese web server is built on UNIX platform. It can manage Chinese text data encoded in different codesets on the same server and provide automatic codeset conversion transparently to client machines when data stored in the server are incompatible with what the client machine can process. The internationalized browser is intended to work under different language and cultural conventions. The Chinese web browser is a localized version of the internationalized web browser whose interface part is developed in an internationalized way. This browser allows users to access the web server either using traditional Chinese codesets or simplified Chinese codeset without the need to match the source document's codeset. Both the web server and web client use the new data type negotiation mechanism of HTTP/1.1 [9] during Internet communication. The development of the Chinese web server is based on the CERN libwww [29]. The internationalized browser is based on NCSA Mosaic written in C language. Its interface part is written on top of Motif under the X windows environment which supports internationalization and localization based on the Locale specifications of ISO POSIX.

To support codeset announcement and automatic codeset conversion for Chinese information access for users who do not have our software, a new approach is devised to complete the same task. In the new scheme, web servers and browsers communicate with each other through the current HTTP protocol which doesn't provide data type negotiation. The data type negotiation is realized through a web page interface and its corresponding CGI(Common Gateway Interface) program. The interface is written in HTML form format asking users to select their preferred codeset by clicking the related button such that the codeset announcement of the client side is completed by users' intervention. This information is then transferred to the remote server via an

HTTP request message sent by the browser, and it is CGI's responsibility to receive the form input data when they reach the server side. The CGI program then tries to investigate the original codeset of the retrieved document and carries out automatic codeset conversion to the retrieved document if needed. Finally, the CGI program announces the codeset of the returned document in the HTTP response message header and the server takes up the job and returns the document to the client at last.

The rest of the thesis is organized as follows. Chapter 2 introduces basic concepts and related work. Chapter 3 presents the design principles and system architecture. Chapter 4 and Chapter 5 describe the design details and implementation of the enhanced server and the enhanced browser respectively. Chapter 6 describes how to realize Chinese information access through normal web server/browser with a new approach - the common gateway interface (CGI). Chapter 7 is the conclusion.

Chapter 2

Basic Concepts And Related Work

The World Wide Web (WWW), or web for short, is one of the most graphical Internet services and is a way of creating a geographically distributed pool of information so that people separated by short or long distances can make information available to others. The web has very strong linking abilities where it allows specially marked words and/or pictures in a document to link/refer to documents which could be of other media and are located in other machines which could be very far away. It is very easy for users to access web information through Internet, just by clicking the buttons. These features contributed the fastest growth and usage of the web. Especially in recent years, more and more web documents written in languages other than English, including Chinese, are available on Internet.

The client-server model is adopted in WWW where a browser can access information on Internet through various Internet tools such as FTP, NNTP, Gopher and HTTP. Figure 2.1 shows the typical client-server model used in WWW. A client is the service requester which is responsible for requesting services from the server. A server is the service provider which is a long living process, referred to as the *daemon process*, waiting to handle requests that may come through network connections [35]. The text data maintained on the server machines are regarded as source web documents. The

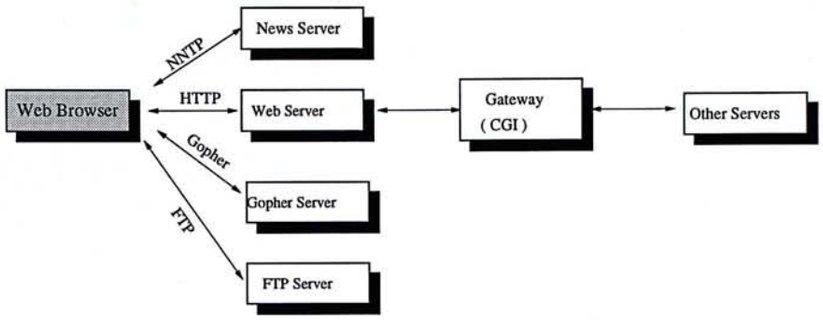

Figure 2.1: Client-Server Model used in WWW

language used by clients and servers to communicate with each other is called protocol. For web servers and web clients (browsers), the protocol they use are HyperText Transfer Protocol(HTTP for short). Currently, HTTP/1.0 protocol is supported by most of the existing web browsers and web servers. HTTP/1.1 is the latest specification which consists of new features of *data type negotiation* mechanism which will be explained later. When clients retrieve documents through a web server, documents accessed are mostly written in HTML language [11]. In this thesis, I confine myself to text document access through web servers only.

In order to fulfill additional language requirements for handling Chinese documents and other non-alphabet based languages, additional features for both HTML language and HTTP protocol must be developed. Since Chinese documents can be written in different codesets, ways have to be found to identify the codeset of documents first, then to exchange this information when clients and servers communicate with each other and finally to carry out automatic codeset conversion if needed.

2.1 Codeset and Codeset Conversion

To fulfill the additional handlement for Chinese information, the first step is to identify the codeset of the retrieved documents. As we know, there are two major character sets of Chinese, both of them are being used widely. One is traditional character set and the other is simplified character set. Corresponding to each character set, there exist multiple codesets. Some of the well-known codesets for Chinese are: GB 2312

Table 2.1: Codesets of Chinese

Character Set	**Country of Origin**	**Codeset**	**Number of Characters**
Simplified Chinese	Mainland China	GB 2312-80	6,763
Traditional Chinese	Taiwan	CNS 11643-1986*	13,051
Traditional Chinese	Taiwan	BIG-5	13,053

for simplified Chinese character set, Big5 and CNS for traditional Chinese character set. Please note the difference between character set and codeset. Character set is just the graphic representation of all characters while codeset is a mapping of a character set to specific numeric codes [31]. Each codeset contains a set of values and each of these values represent one character with absolutely no duplications. When handling Chinese characters, computer systems deal with the numeric code of each character defined by a certain codeset rather than the related character set.

As an ideographic language, Chinese has literally thousands of characters. Because of this, an 8-bit codeset with maximum of 256 characters obviously is inadequate. In fact, the three major codesets for Chinese - GB 2312, Big5 and CNS, all adopt 16 bits (two bytes) to represent each Chinese character. Table 2.1 [26, 27, 32] lists the information of the three major codesets for Chinese. Here the number of characters refers to the number of hanzi characters in the corresponding codeset excluding other alphabet characters in the same codeset since the large number of hanzi characters are under our main concern. Note that CNS 11643 was revised in 1992 to increase its characters to 48,228 [31].

Because of historical reasons, the two major Chinese character sets are not compatible and the traditional character set is a super-set of the simplified one. The code range of different codesets is different and has overlap in some extent. Table 2.2 [26, 27, 32] illustrates the code range of all hanzi characters in different codesets. Note that in the above table, the code range of CNS 11643 just denotes the range of all hanzi characters in plane one of CNS 11643. For plane two or more, hanzi characters are represented by four bytes instead of two bytes which are more complicated than the

Table 2.2: Code Range of Hanzi Characters in Different Codesets

Codeset	First Byte Range(Hex)	Second Byte Range(Hex)
GB 2312-80	B0 - F7	A1 - FE
BIG-5	A4 - C6, C9 - F9	40 - 7E, A1-FE
CNS 11643*	C4 - FD	A1 - FE

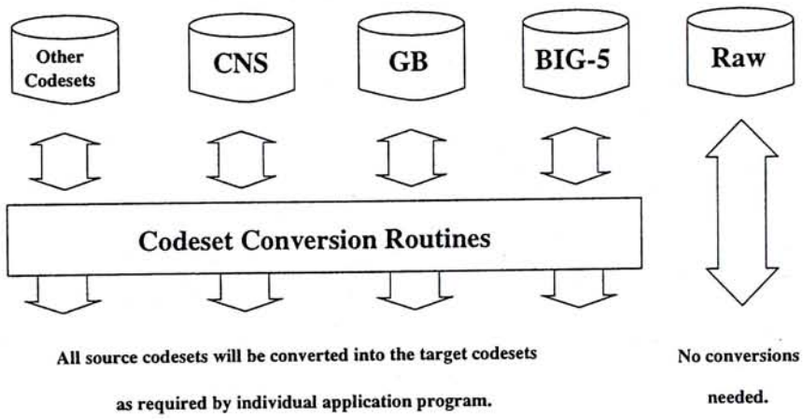

Figure 2.2: Codeset Conversions

two-byte system. From Table 2.2, it is easy to find that one code value can represent different Chinese characters in different codesets. For example, code value 0xB0AE is '爱' in GB 2312-80, but it is '乾' in BIG-5. This brings the incompatibility. The co-existence of multiple codesets is the barrier for people from Hong Kong, Taiwan and Mainland China to exchange Chinese information and causes the unnecessary duplication of various versions of documents or softwares for different codesets. This brings us the need of codeset conversion.

Codeset conversion is the procedure to translate a document from one codeset to another. For the major three codesets of Chinese, we need at least converters for pairs of GB/BIG5, BIG5/CNS and CNS/GB. Figure 2.2 [13] illustrates the codeset conversion between different codesets. Because different codesets are not compatible, there are 1-to-N and N-to-1 mappings instead of only 1-to-1 mapping for the characters being converted when translating them from one codeset to another. The simplest way is to choose one of all mapping characters as a default output no matter it is correct or not in the context. Another way is to provide all possible mapping characters, and let users to choose. The best way is to choose the right one intelligently by the converter.

This needs the involvement of semantic analysis in context. Currently, we use the first method in our codeset routines.

2.2 HTML Language

Web documents are mostly written in a language called HyperText Markup Language (HTML) [11]. HTML is designed to specify the logical organization of a text document, with important extensions for hypertext links and user interaction. HTML requires you construct documents with sections of text marked as logical units, such as titles, paragraphs, or lists, and leaves the interpretation of these marked elements up to the browser displaying the document [11]. The structure of an HTML document can be marked by various tags. Here is a simple example:

```
<html>
  <head>
    <title> Demonstration of HTML Document </title>
  </head>
  <body>
     Hello World!
  </body>
</html>
```

Each HTML document starts with the tag <HTML> and ends with </HTML>. In most cases, an HTML document has a head part and a body part which are marked by tag <head> and tag <body> respectively. The above example shows that this document has a title named 'Demonstration of HTML Document' and the body is 'Hello World!'. Figure 2.3 shows the corresponding web page of the above HTML file under Netscape 3.0.

As Chinese web documents can be written in different codesets, and for the purpose of fulfilling additional requirements for Chinese information access, ways have to be found to identify their codeset, which is referred to as *codeset identification.* Earlier versions of HTML have no mechanism to tell data are written in what codeset, everything defaults to ISO-8859-1 which was designed for European languages [42]. Even

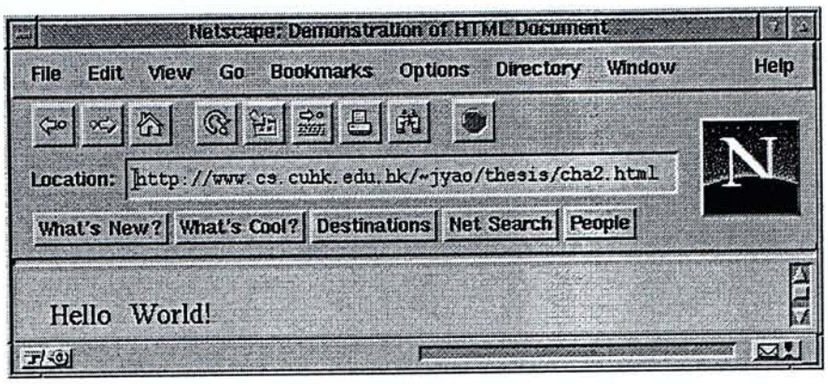

Figure 2.3: The Web Page of a Simple HTML Document

with the only codeset available, HTML has still been used to describe data in other languages and encodings at the expense of interoperability [42]. In other words, HTML documents written in other codesets such as Chinese are forced to be interpreted as ISO-8859-1. HTML version 3.0 [34] has included a new language tag <LANG> to announce the codeset of text data. For a multi-part document, each pair of <LANG> tags denote a certain language or codeset for that part. Therefore it is easy to indicate the content type information of a multilingual or multi-codeset mixture in one document with several <LANG> tags. Here is an example:

```
            Single Web Document
-----------------------------------------
<html>
<title> Multiple Codeset Document </title>
<LANG=big5>
...繁 體 字 寫 成 的 部 分...
</LANG>...
<LANG=gb2312>
...简 体 字 写 成 的 部 分...
</LANG>...
</html>
-----------------------------------------
```

With the help of tag <LANG>, it is easy for both the web browser and web server to identify the codeset of the retrieved documents. However, no commercial browsers and servers are able to interpret this new tag yet.

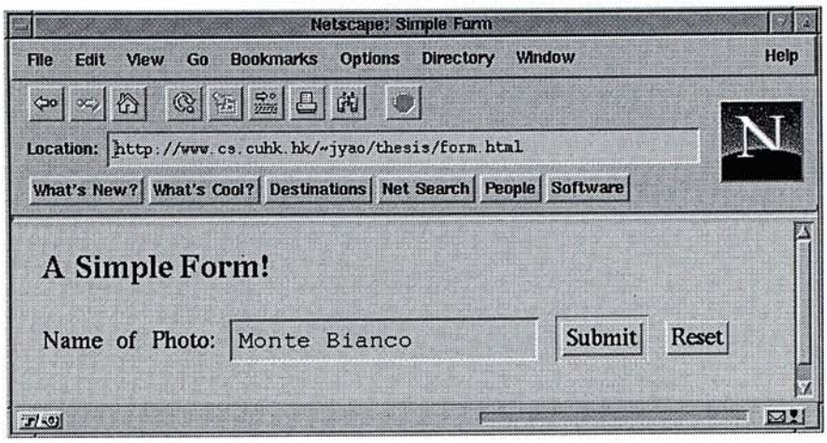

Figure 2.4: The Web Page of a Simple Form

Another powerful feature of HTML language is that it provides fill-in forms to realize the interaction with users. The tag <form> makes HTML documents' authors establish interactive interface, collect input data from users and make response based on users' input. This makes the web documents alive instead of being static all the time. Here is an example:

```
<form action="http://www.cs.cuhk.hk/cgi-bin/photo">
    Name of Photo: <input type="text" name="photoname" size=30>
    <input type="submit" value="Submit">
    <input type="reset"  value="Reset">
</form>
```

Figure 2.4 shows the web page corresponding to the above example. In this example, the first line ties the data of the form to a particular program (photo) on the indicated HTTP server *(http://www.cs.cuhk.hk)* at the directory *"/cgi-bin"*. The second line asks users to input the name of the photo they want to see. The third and fourth lines are designed for *Submit* button and *Reset* button respectively. If there are mistakes in users' input, they can click *Reset* button to clear all input data, otherwise, after they click *Submit* button, the form data (Here are "Monte Bianco": the name of the photo) will be sent to the program (photo) and the corresponding photo will be sent back later.

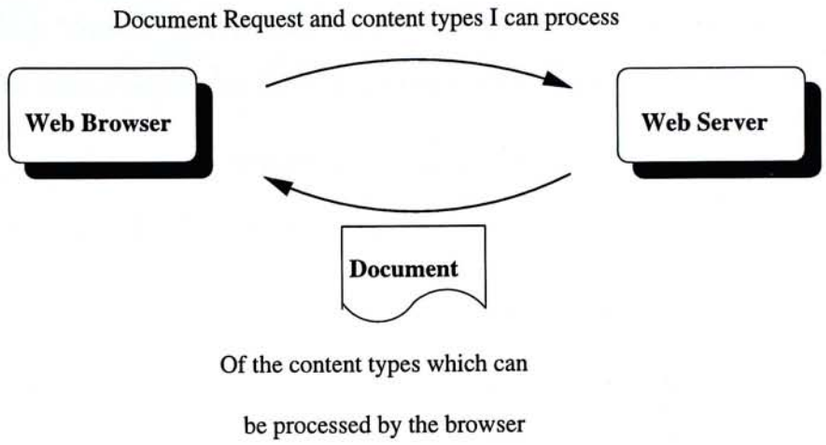

Figure 2.5: Negotiation of Data types between Web Browsers and Web Servers

2.3 HTTP Protocol

The communication between a web browser and a web server is carried out by Hyper-Text Transfer Protocol (HTTP) [9, 21]. HTTP is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management. HTTP protocol has been in use by the World Wide Web global information initiative since 1990 [21], and most of the existing web servers and web browsers adopt pre-HTTP/1.1 protocol, such as HTTP/1.0 [21] in most cases.

The HTTP protocol is based on a request/response paradigm. A requesting program (termed a client) establishes a connection with a receiving program (termed a server) and sends a request to the server in the form of a request method, URL, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content. The server responds with a status line, including its protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible body content [21]. The feature of HTTP is the typing and negotiation of the data representation, allowing systems to be built independently of the data being transferred [9]. The negotiation of data types between web browsers and web servers is shown in Figure 2.5. The web

browser sends a request telling what kind of data types it can handle, the web server sends back the document with the data type which is what the browser can support.

Communication through HTTP protocol is 8-bit clean which ensures safe transmission of all forms of data including Chinese [11]. An HTTP connection has four steps:

1. Open connection — The client contacts the server at the Internet address and port number specified in the URL (the default port is 80).

2. Request for service — The client sends a message to the server, requesting service. The message consists of HTTP request header defining the method requested and the information about the capabilities of the client. The header is followed by the data being sent to the server (if any). Typical HTTP methods are `GET`, for getting an object from a server, or `POST`, for posting data to an object on the server.

3. Response from server — The server sends a response to the client with response header describing the state of the transaction (for example, the status of the response — successful or not — and the type of data being sent), followed by the actual data.

4. Close connection — The connection is closed.

HTTP messages consist of requests from client to server and responses from server to client. In HTTP/1.0, messages are Full-Request and Full-Response. The grammar of them are as follows:

```
Full-Request    = Request-Line
                  *( General-Header
                  |  Request-Header
                  |  Entity-Header )
                  CRLF
                  [ Entity-Body ]
Full-Response   = Status-Line
```

```
*( General-Header
|  Response-Header
|  Entity-Header )
CRLF
[ Entity-Body ]
```

Both the request and the response consist of three parts: the first line of the message, the header and the body. Request-Line begins with a method, followed by the URL requested and the protocol version, for example:

```
GET http://www.cs.cuhk.hk/Index.html HTTP/1.0
```

This line means to retrieve document at *http://www.cs.cuhk.hk/Index.html* with protocol *HTTP/1.0*. The Status-Line consists of the protocol version followed by a code number and its associated textual phrase. For example:

```
HTTP/1.0 200 OK
```

This means the server uses HTTP/1.0 as the communication protocol and the status code is 200, standing for success, phrase 'OK' has the same meaning.

The General-Header includes information of Date, Mime-version, and Pragma which decides whether a cache is chosen. The Request-Header of a Full-Request allows the client to pass additional information about the request, and about the client itself, to the server, such as: User-Agent, Authorization and so on [21]. Similarly, the Response-Header of a Full-Response allows the server to pass additional information about the response as well as the server itself to the client [21]. The Entity-Header in either the Full-Request or the Full-Response defines optional information about the trailing Entity-Body, if no body is present, about the resource identified by the request. Here is the Entity-Header fields defined in HTTP/1.0 [21]:

```
Entity-Header  = Allow
               | Content-Encoding
               | Content-Length
               | Content-Type
```

```
| Expires
| Last-Modified
| extension-header
```

The field Content-Type of Entity-Header in HTTP/1.0 provides simple data type negotiation in one direction: from the server to the client. Codeset information can be indicated in this field by charset parameter to tell the client what kind of data type the retrieved document is, for example:

```
Content-Type: text/html; charset="ISO-8859-1"
```

Although this Content-Type header field can provide codeset information about the trailing data, in general, most of the existing web servers do not provide charset parameter in the Content-Type field when they send back response message to clients. Even though web servers do provide charset information in this header field, few web browsers know how to interpret it, instead, just ignore such information when analyzing the Content-Type header. In addition, since ISO-8859-1 is the default character set (or codeset) in the case of HTTP protocol [21], all text data in the Entity-Body are forced to be interpreted as written in ISO-8859-1, misinterpretation of Chinese text data is unavoidable.

In general, data contained in the response message's Entity-Body are mostly simple ASCII text(or 8-bit clean) written in HTML language. We are particularly concerned about the codeset in which the HTML document(the body data) is written. Consider the following scenario. Suppose a simplified Chinese source document on the server is coded in GB2312 and suppose that the client wishes to read the document in traditional Chinese, then the document must be converted to a codeset that represents traditional Chinese, say Big5, before it can be displayed on the client browser. This brings out the Chinese specific requirement of codeset conversion support.

Before codeset conversion is carried out, ways have to be found to tell what codeset in which the web document is written. As discussed in Section 2.2, the codeset tagging for web documents can be done through the new tag <LANG> defined in HTML

version 3.0. In order to minimize codeset conversion efforts, and to provide flexibility of codeset conversion at either the client side or the server side, the client and server must announce information on the codesets they can support, referred to as *codeset announcement.* For instance, if the source document is written in GB2312, and the client side recognizes GB2312, no codeset conversion would be needed on the server's side if the server is aware of such information. As another example, suppose the client side only supports ISO-8859-1 and Big5 encodings, the codeset conversion for a source document written in GB2312 can only be done on the server side provided that server knows what codeset to convert the document to.

The new features of HTTP/1.1 [9], use the basic idea of language labeling facility in MIME [2] to allow the negotiation of data types between the server and the client. It is the HTTP header which provides us with the information regarding the data types (including language, codeset and encoding etc.) of documents which we are retrieving. From HTTP/1.0 [21] onwards, whenever a client requests a web document from a server, the client will pass along a list of data types which it can support. In receipt of such a request, the sever will try its best to send HTML document with only the data types supported by the client. The header of the returned data shall indicate which data type of the trailing content is. This is subsequently interpreted by the client. The HTTP/1.1 header contains two fields related to our discussion, namely, `Accept-Charset` and `Accept-Language` in the HTTP request message and `Content-Type`(with *charset parameter*) and `Content-Language` in the HTTP response message. Their formats are:

```
*Accept-Charset  = "Accept-Charset"  ";" 1#charset
*Accept-Language = "Accept-Language" ";"
                   1#(language-tag[";" "q" "="qvalue])
*Content-Type    = "Content-Type"    ":" media-type
                   media-type = type "/" subtype *( ";" parameter )
                   type       = token
                   subtype    = token
                   parameter  = attribute "=" value
                   attribute  = token
```

```
                    value       = token | quoted-string
*Content-Language = "Content-Language" ":" 1#language-tag
```

The first two `Accept` fields provide the codeset announcement mechanism for the client. The latter two `Content` fields are filled up by the server. With the help of these four fields, the client sends the request with the codeset announcement information to the server, according to this codeset preference information and based on the codeset of the source document retrieved by the client, the server decides whether automatic codeset conversion should be carried out. According to the data type specification given by the server, the client can interpret the data properly.

2.4 I18N And L10N

As a Chinese Internet access tool, the client interface must be customized in a way that is convenient for Chinese users. For example, the client interface for people in Mainland China should be customized to simplified Chinese. However, it is inflexible and unnecessary to set up several versions of the browser for different codeset environments, such as having different versions for simplified Chinese and traditional Chinese. Since the core part of the software is the same, the browser design should be independent of language and culture conventions. This design methodology is known as internationalization (I18N) [22, 31]. With an internationalized browser, each time it is invoked, the specific information on the supported language, cultural data and codeset is plugged into the program dynamically. This process of preparing data for a particular environment is called localization (L10N) [22, 31].

To facilitate I18N and to make L10N easy, ISO POSIX provides several mechanisms. One is the introduction of *Locale* [22, 31]. Locale is defined to hold the collection of rules and text specific to a language and geographical area, such as collation rules if different from the internal code sequence, date and time formats, and character classification data for a language or culture such as alphabet letters, upper/lower case letters, and control characters, etc. Another mechanism is the *message catalog* system

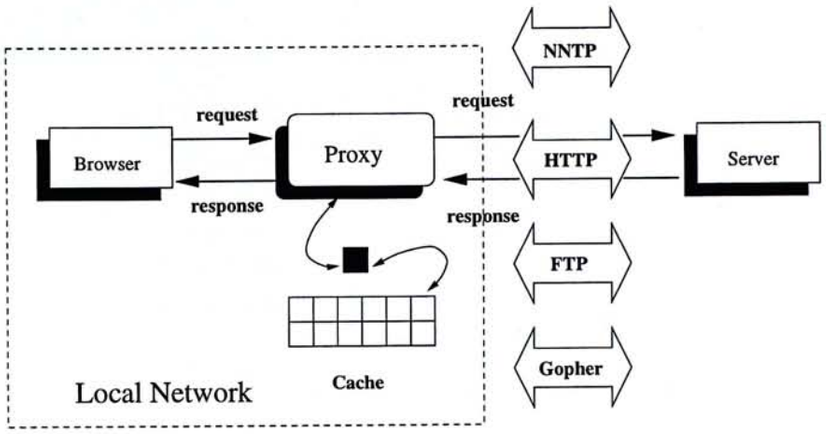

Figure 2.6: Proxy server act as a client and a server

[22, 31]. A message catalog contains program context data such as prompt information, help messages, error messages, etc.. By separating this information from the program, the program can be independent of the operating environment and this set of data for different languages can be prepared in a uniformed way. When a browser is invoked under a certain Locale, the messages related to this particular Locale can be extracted from the message catalog automatically.

2.5 Proxy Server

In general, browsers connect with servers directly to retrieve HTML documents. However, in some cases, clients are connected with a so-called *proxy server* [28] which is in turn connected to the server. A WWW proxy server, *proxy* for short, provides access to the web for people on closed subnets who can only access the Internet through the firewall machine. A proxy is a special server which typically runs on a firewall machine. It waits for a request from inside the firewall, forwards the request to the remote server outside the firewall, reads the response and then sends it back to the client [11]. It is possible to arrange the same proxy to be shared by all the clients within a given subnet so as to perform caching of documents that are requested by a number of clients. Figure 2.6 illustrates the proxy server's architecture.

Nowadays many web browsers have the proxy support built in. The proxy acts as

Table 2.3: Language encodings as supported by Netscape 3.0.

Language	*codeset*
Western	(Latin-1)
Central European	(Latin-2)
Japanese	(Auto-Detect)
Japanese	(Shift-JIS)
Japanese	(EUC-JP)
Traditional Chinese	(Big5)
Traditional Chinese	(EUC-TW)
Simplified Chinese	(GB)
Korean	(EUC-KR)
Korean	(ISO-2022-KR)
Cyrillic	(KOI8-R)
Cyrillic	(ISO 8859-5)
Greek	(ISO 8859-7)
Turkish	(ISO 8859-9)
User-Defined	

both a client and a server in the whole transaction since the initial document requests generated by the browsers. When requesting for a document, on the browsers' side, the proxy is a server to receive the requests from them; and on the other side it will inspect the browsers' messages, filter/process them before forwarding it to the target server. This is also true for the other direction when the proxy receives the returning document from the target server and the proxy will perform proper processing on them before delivering it to the browsers. In case the server where the source document is located is not enhanced to support Chinese specific requirements, an enhanced proxy server can be set up as the bridge between clients and target servers.

2.6 Related Work

Currently, most of the web servers and browsers do not support interpretation of HTTP/1.1 and HTML/3.0. However, some do provide document encoding options for users to switch manually if the document does not match the viewing browser's current encoding. In Netscape 3.0, for example, users can select one of the supported codesets as shown in Table 2.3.

Table 2.4: Access result using Netscape and CMosaic.

Codeset of Source Document	**User Preferred Codeset Under Netscape 3.0**	**User Preferred Codeset Under CMosaic 2.4**
明 报(Big5)	Document-encoding: Big5: OK EUC-TW: OK GB: Not work	Font: Chinese(gb2312): Not work Chinese(Big5): OK Chinese(HZ): Not work
华夏文摘 (GB)	Document-encoding: Big5: Not work EUC-TW: Not work GB: OK	Font: Chinese(gb2312): OK Chinese(Big5): Not work Chinese(HZ): OK
华夏文摘 (Big5)	Document-encoding: Big5: OK EUC-TW: Not work GB: Not work	Font: Chinese(gb2312): Not work Chinese(Big5): OK Chinese(HZ): Not work

Here is an example when users use Netscape 3.0 or CMosaic (Mosaic-l10n) [38, 40] to access Chinese Information through Internet. The operation procedures are nearly the same when using both browsers. At the beginning, users may choose the font from the menu, and then access Chinese web documents encoded in different codesets. Table 2.4 shows the access result.

From Table 2.4, it is obvious to find that only when the codeset of the source document is identical to the one chosen by users at the client side, or in other words, only when users just choose the right codeset identical to that of the required document can the browser display the information correctly, otherwise what we will see are garbage data. In most cases, users of Netscape 3.0 are assumed to know the codesets of the target documents they are retrieving, which seems not to be a good idea, as in many cases the codesets of the target documents are being identified by the HTTP header returned. There is no automatic codeset conversions built in, therefore even when Netscape 3.0 knows the codeset of the retrieved document, it will not perform any extra processing except passing directly to the users. As a result there are cases where the users have set a particular language encoding (either as a default or setting it via the option menu), say GB 2312 , and requesting to view a document written in another

codeset, say Big-5. Netscape 3.0 will dutifully retrieve the requested document, but will not perform any codeset conversions on it. Using the existing fonts for GB 2312, it is obvious that the browser will not be able to show the right shapes of characters for the target documents. In that case the users will be forced to choose the proper Document Encoding from the menu and the characters will be re-displayed in their right shapes. This requires users' interventions. And in the cases where the codeset of the retrieved document is not known, it is tedious for users to guess it correctly. Therefore what the current browsers can provide is not as what users anticipate.

To solve this problem, we should enhance the web browser so that the preference of codeset for viewing the document may be selected by users dynamically and no matter what codeset the required document is encoded in, the browser can display it in the codeset as what users expect and users will not be aware of the automatic codeset conversion , if any, at all.

Chapter 3

Design Principles And System Architecture

The objective of our system is to provide a friendly environment for users to fetch, exchange and process Chinese text data via Internet. To realize our main goal, the core idea is to provide *codeset announcement* between web servers and web browsers and to carry out *automatic codeset conversion* if the original codeset is incompatible with what users require or prefer. Since our development is based on the current commonly-used web server and browser, I first explain the technology used by most existing web systems, then describe the new approach devised in our web system, at last I discuss the overall system architecture of our web system.

3.1 Use of Existing Web System

3.1.1 Protocol

The protocol which most of the current web servers and browsers adopt is HTTP/1.0 [21] which doesn't provide sufficient *data type negotiation.* As introduced in Chapter 2, HTTP messages consist of requests from client to server and responses from server to client. In HTTP/1.0, there is no header field of a request message which can be used

to announce the codeset that the browser can support. Also, in the response message, there is only one header field named *Content-Type* which is defined to indicate which codeset the trailing data are encoded in. Although this Content-Type header field can provide codeset information about the trailing data, in general, most of the existing web servers do not provide charset parameter in the Content-Type field when they send back response message to clients. Even though web servers do provide codeset information in this header field, few web browsers know how to interpret it, instead, they just ignore such information when analyzing the Content-Type header. In addition, since ISO-8859-1 is the default codeset in the case of HTTP protocol [21], all text data in the Entity-Body are forced to be interpreted as written in ISO-8859-1, misinterpretation of Chinese text data is unavoidable.

Comparing with HTTP/1.0, HTTP/1.1 [9] improves much on its data type negotiation mechanism. The codeset announcement can be done in two directions. For each HTTP request, two new Request-Header fields are provided. They are Accept-Charset and Accept-Language [9]. With the help of these two fields, the client now is able to pass the local environment (the language and codeset the client can support) to the server. This completes the data type negotiation from the client to the server. On the other hand, HTTP/1.1 defines a new Entity-Header field Content-Language which tells what language of the text data in Entity-Body is. With the help of both Content-Language and Content-Type (with charset parameter) Entity-Header fields, the server can tell the data type of the retrieved document when it sends back the HTTP response to the client. This completes the data type negotiation from the server to the client.

To support codeset announcement, HTTP/1.1 is adopted as the protocol communicated by our Chinese web server and browser with each other so that two directions' data type negotiation can be realized. Furthermore, using the new headers fields defined in HTTP/1.1 makes the data type negotiation applicable to many other languages/codesets , not just Chinese. It is sure that HTTP/1.1 is the trend for the web world and its new features make the access of web documents written in various languages and codesets become much more easy and convenient.

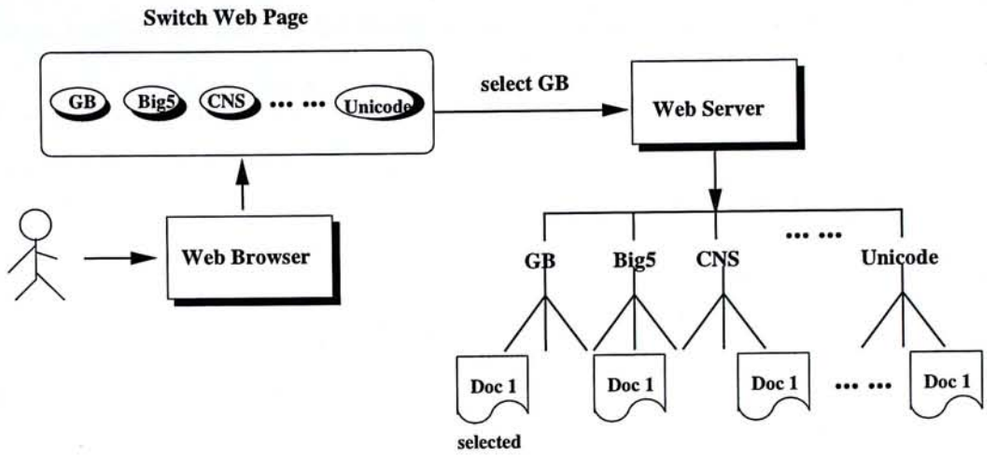

Figure 3.1: File Storage Structure on Multi-codesets Web Servers

3.1.2 Avoid Duplication of Documents for Different Codesets

Most of the English-based or alphabet-based language web servers provide only one version document over the Internet. For example, the homepage of Netscape has only English version information. For documents written in ideographic languages, such as Chinese, some web servers may provide multiple versions for the same document with different codesets. For example, the home page of *Hua Xia Wen* [6] on a web server located in U.S.A, provides multiple versions for the same document: GB (simplified Chinese), Big5 (Traditional Chinese), HZ, etc. Users just click a certain button related to a particular codeset to see the right version of that document.

The idea for the existing web servers supporting multiple codesets is to adopt the directory tree structure concept which keeps the documents with various codesets under different sub-directories [36]. For example, the documents encoded in Simplified Chinese are stored under the sub-directory GB, while the documents encoded in Traditional Chinese are stored under the sub-directory BIG5 and CNS respectively depending on various systems's conventions. Figure 3.1 illustrates the file storage structure on multi-codesets web servers. When a web browser sends a request to a multi-codesets web server, the server tries to fetch the retrieved document with the codeset selected by the web browser from the corresponding sub-directory. At the browser side, the selection of different version document can be done through clicking

a button, a particular part of an imagemap or a hyperlink in a switch page.

The approach of storing documents with multiple codeset versions has many disadvantages [36]:

1. If the web server supports N codesets, it has to maintain N copies of documents with the supported codesets. This requires quite a lot of effort to convert each document from one codeset to another.

2. It wastes a lot of disk space storing duplicated information.

3. It is difficult for the management and is liable to cause inconsistency of different version documents. For instance, once one version document has been updated, all other copies should be modified timely, otherwise inconsistency will happen.

4. Current operating systems usually support only one kind of codeset which makes it hard for web clients to support multiple codesets.

To overcome the limitations of the above approach, some web servers use image files instead of the text equivalent to fulfill the requirements for web clients which don't support the codesets provided by the multiple-codesets web server. However, image files usually occupy a huge disk space which seems not to be a good solution, new facilities should be added into web systems.

In our design, with the help of *codeset announcement mechanism* and *automatic codeset conversion facility*, our web server stores only one version for each Chinese document, either in traditional Chinese codesets or in simplified Chinese codeset. There is no need to maintain other versions for other Chinese codesets at the same time. Each time, when our browser tries to access a Chinese document on our server, the server is capable to decide whether codeset conversion needs to be done to convert the retrieved document or not. If the original codeset of the retrieved document is incompatible with what the browser can support, automatic codeset conversion will be conducted. At last, the converted document will be sent back to the browser for display.

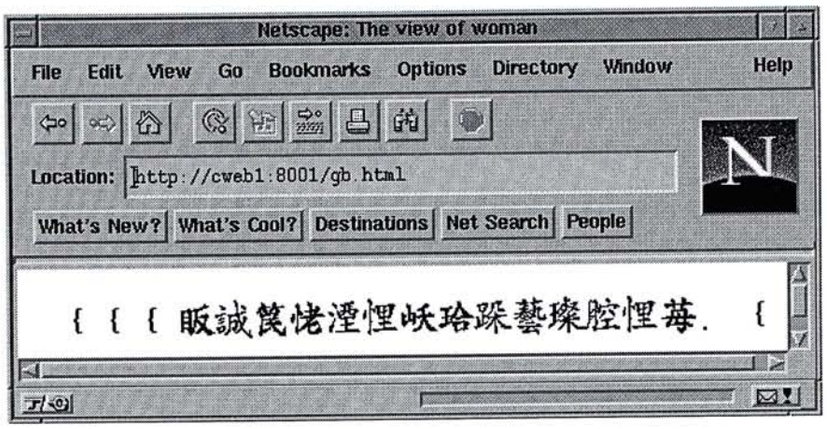

Figure 3.2: Output for viewing a Chinese GB web page using a Big5-based browser

3.1.3 Support On-line Codeset Conversion Facility

It is very common that only one codeset is supported on all computer platforms in a certain region or country. For instance, nearly all computer systems used in Mainland China only support simplified Chinese - GB codeset, in the meanwhile, most of the computer systems used in Taiwan and Hong Kong support traditional Chinese with codeset Big5, and some of them also support CNS, however, they do not support simplified Chinese at most of the time. Suppose a PC-platform web browser Netscape 3.0 only supports Big5 encoding on its local machine, although users can select "Document Encoding" from Netscape's menu to different codesets, users still cannot read documents encoded in GB in this case, because there is no resource for simplified Chinese, such as fonts, at the client machine. Figure 3.2 shows the access result of the example. In this example, although some Chinese characters are displayed on the screen, the sentence as shown in Figure 3.2 is meaningless due to misinterpretation of the text data. One of the solution is to download the documents encoded in codesets other than the one supported by the local machine, and use other codeset conversion softwares to convert them later. This brings inconvenience to users as it gives unnecessary disk space burden for users to save the intermediate documents on their local machine, and wastes their time to do the conversion by themselves, and will frustrate them to read such documents in the future again. It is unfavorable to information exchange among people in different regions and countries.

Let's see another situation. Suppose the local machine supports many codesets on its own, and users want to access a web document encoded in a codeset which is not known by users in advance. If users happen to set the current 'Document Encoding' of the web browser compatible with the one of the retrieved document, there is no problem for them to read the document. However, it may be the case that the two are incompatible, for example, users set the current display environment to simplified Chinese, it is very possible that the document is encoded in traditional Chinese, in this case, garbage data will be displayed on the screen like shown in Figure 3.2, users are forced to try different 'Document-Encoding' manually which is very tedious, and if the number of codesets increases, the manual intervention will be time-consuming and boring.

What we have learned from the both cases leads us to the consideration of on-line automatic codeset conversion. With on-line codeset conversion, users can access documents encoded in different codesets without the need to worry about the incompatibility between codesets supported by the web browser and that of the retrieved document, all conversions are conducted transparently and automatically.

3.1.4 Provide Internationalized Interface of Web Browser

Most of the existing web browsers provide English interface for users. As a Chinese information access tool, the browser interface should be customized to support Chinese so that users can operate the web browser in their familiar environment, either in traditional Chinese or simplified Chinese.

The traditional method to realize the objective is to add more codes to the original browser program, so that the modified browser supports bilingual environment. However, this method has some drawbacks. As the number of Chinese codesets supported increases, a large amount of repetition work will be done to modify the original code of the program to support the new language/codeset. Also the program is not extensible and it is difficult to guarantee the consistency with different codesets/languages. As the supported codesets grows, so does the code size which causes some perfor-

mance degradation. The new approach named Internationalization (I18N for short) is adopted in our design. The main idea of I18N is to separate the logical control of software from the data it handles. The software includes calls to generalized routines that get language- or culture-specific data at run time. The data still has to be localized. An internationalized system is capable of supporting new languages and cultures without changes to the source code of the program.

In our system, all program context messages are isolated from the program, and they are stored in *message source files* under different directories for different codesets. Each message source file has identical format, only the content itself is different from one to another. Each time, when users invoke our browser, there are some ways for them to set the local environment. After the environment has been set up, the corresponding messages are plugged into the program when it runs. With I18N approach, adding a new language/codeset is quite straightforward, just by adding a new message source file under related directory. There is no need to change the source code of the program at all.

3.2 Our Approach

To solve all problems of Chinese information access through the web, we proposed a new approach to enhance the current web system. The main goal of our web system is to provide codeset announcement [13] for both Chinese Internet documents and for browser environments so that automatic codeset conversion can be provided transparently. This allows a browser to view a document in the user's preferred codeset regardless of the codeset of the source document being retrieved. In brief, we first need language/codeset tagging in the web documents using HTML 3.0. Secondly, we must enhance servers and browsers(or proxies) by supporting HTTP/1.1 so that data type negotiation can be carried out. Thirdly, we must add on-line codeset conversion functionalities [14] into the enhanced server and browser. Currently, most servers and browsers do not support data type negotiation and on-line codeset conversion, so we

have to discuss the different cases and their solutions.

3.2.1 Enhancing the Existing Browsers and Servers

The enhancement of browser and server consists of two parts: one is supporting HTTP/1.1, the other is providing automatic codeset conversion functionality. If both sides, the browser and the server support HTTP/1.1, the codeset announcement is straightforward. However, there are still browsers and/or servers which are running pre-HTTP/1.1 protocols. Therefore, different situations need to be discussed further.

Case 1: Having An Enhanced Browser Only

In this case, only the browser is enhanced, therefore data type negotiation between the browser and the server cannot be carried out, and automatic codeset conversion can be done on the browser side only if the retrieved document is written in HTML/3.0 with language tagging. Since the browser is invoked by an individual user, caching of converted documents and sharing them by other users are difficult. In other words, even if the same original document is retrieved by N users from the same network and assuming that they are all requesting for the same target codeset, N retrievals are still needed and N codeset conversions must be done, which waste network bandwidth as well as system resources. It is also possible that a given browser does not have the required converters. In that case the original document is sent back to the browser without any change.

Case 2: Having An Enhanced Server Only

Using a normal browser and an enhanced server, the server should convert the required document into one which is acceptable by the browser. However for browsers preceding HTTP/1.1, the server will not know which codeset(s) the client can handle. It is because the client will not pass the codeset information in the header. Although the server can perform codeset conversions, it doesn't know the target codeset as required by its client.

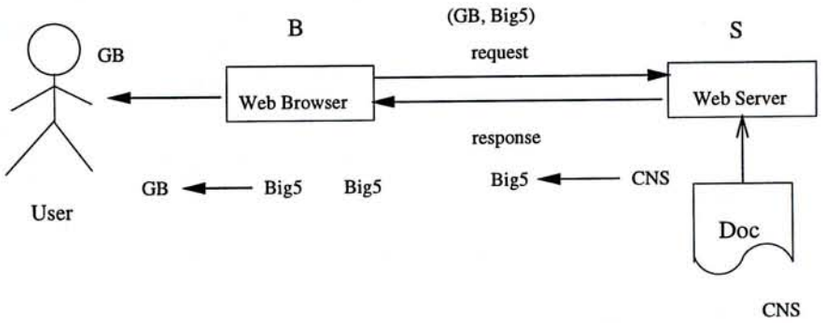

Figure 3.3: Multi-Codeset Conversions of Both Web Browsers And Web Servers.

Case 3: Having Both Enhanced Server And Enhanced Browser

There are cases where we must have both the enhanced browser and the enhanced server. Consider the scenario shown in Figure 3.3. where a browser B can accept codeset GB and it can perform codeset conversion from codeset Big5 to GB. Assume that a user chooses to view document in GB, and the document requested by B is written in codeset CNS. Also assume that the server S servicing the requests can convert from CNS to Big5. Now when B requests for the document from S, it specifies that it can accept codesets GB as well as Big5. Obviously S cannot simply pass the document in its original codeset (i.e. CNS), otherwise the browser would not know how to process it. In receipt of the request, the server knows that the browser can accept both GB and Big5, Therefore, the server can perform CNS-to-Big5 codeset conversion. In doing so, the server informs the browser that the returning document is in codeset Big5. This can be done via HTTP/1.1 header. Upon receiving this header, the browser can perform another codeset conversion — Big5-to-GB. The final characters displayed to the user will be in GB which is what was required in the first place. Ideally, if all the browsers and servers can be updated to HTTP/1.1, codeset conversions would be quite straightforward. The server will simply look at what the client can accept and try to convert the original document into one which is acceptable by the client.

However, to deal with servers which still use pre-HTTP/1.1, even an enhanced browser states what codeset it accepts, the server can not react to this option. Since the target server is beyond users' control, we must devise a way to do conversions when dealing with pre-HTTP/1.1 servers. This brings us to use enhanced proxies to deal

with such a case.

3.2.2 Incorporating Proxies in Our Scheme

As mentioned in the previous section that it is not possible to perform automatic codeset conversions for pre- HTTP/1.1 servers, we can add an HTTP/1.1 proxy server in the path between a pre-HTTP/1.1 server and an enhanced browser. Instead of asking the browser to contact the target server directly, the browser can contact the proxy server first. Effectively the proxy server acts as a client to the remote target server on behalf of the browser.

When the proxy receives a request from a browser, it inspects the header and notices the codesets acceptable by the browser. It is necessary for the proxy to be equipped with various codeset converters, so that it may perform the codeset conversions before returning the document to the browser. Before forwarding this request to the target server, the proxy may add any extra codesets acceptable by it to the header, so as to increase the chance of getting the document in one of the codesets as acceptable either by the proxy or the browser. Although the target server might not be able to pass codeset information in the header, the proxy can still parse the document if it is written in HTML/3.0 and try to identify its codeset. In principle, the codeset information will be available only for HTTP/1.1 or post servers. If such information is available, there is no need for the proxy to search for this information again in the document.

Having identified the original codeset and the target codeset, the proxy can perform codeset conversions before returning the document to the browser. It is possible to configure the proxy so that a certain degree of caching will be enforced. With the presence of the cache facility, the conversion process can be avoided if the same document is accessed again when the converted document in the proxy's cache shall be returned immediately. The browser can get an extremely fast response from the proxy, although the user may perceive that the document is sent directly from the target server. If we can ignore the exception situations possibly occurred during the codeset

Table 3.1: Implementation Possibilities.

Browser	**Proxy Server**	**Remote Server**	**Work or Not?**
HTTP/1.1	Optional	HTTP/1.1	Yes
HTTP/1.1	Mandatory	Pre-HTTP/1.1 (Post-HTML/3.0 documents)	Yes
Pre-HTTP/1.1	No	HTTP/1.1 or Pre-HTTP/1.1	No

conversion process, the user may also think that all the documents are written in the codeset as specified by him/her.

Conceivably the proxy and the browser should reside on the same sub-net, this is to ensure the relatively quick response and easy management from the proxy. The installer of the proxy and/or the browser should install the codeset conversion libraries [12] for both applications. It is also clear that the proxy works even for HTTP/1.1 browsers which cannot perform any codeset conversions. Table 3.1 illustrates the possibilities of implementation. From Table 3.1, we can see that if both the browser and server are HTTP/1.1 compatible, whether there is an enhanced proxy or not between the browser and the server, they are sure to work fine. If the server only supports Pre-HTTP/1.1, the system can only work when the browser is enhanced, and there must be an enhanced proxy between the browser and the server, and the HTML document is written according to HTTP/3.0 specification. Otherwise, the proxy cannot carry out automatic codeset conversion if the retrieved document is not written using the <LANG> tag of HTML/3.0 as the normal server doesn't provide any data type information of the retrieved document at all and there is no way for proxy to identify the codeset. However, if the browser is not enhanced to be able to send its local environment to the server, no matter what kind of proxy and the remote server are, the system doesn't work.

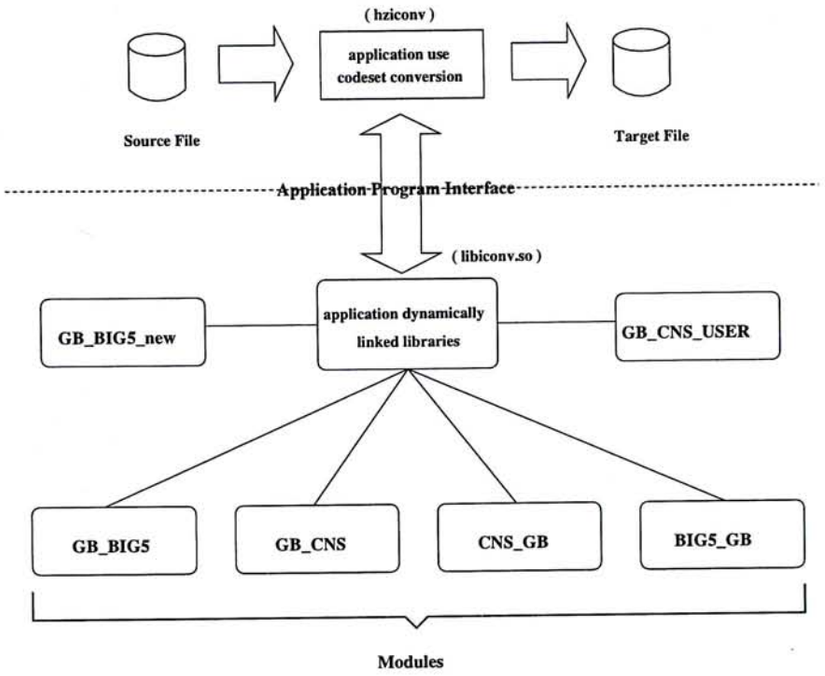

Figure 3.4: The Architecture of API for Codeset Conversions

3.2.3 Automatic Codeset Conversion

Automatic codeset conversion is the important part of our new web system. Codeset conversion routines for different codesets of Chinese have been developed in the Hanzix Open Systems [12, 14]. A uniform Application Program Interface (API) is provided for individual application to do the codeset conversion from one codeset to another. The architecture of the API is shown in Figure 3.4.

The Codeset Conversion Application Program Interface can be seen as a set of functions which are callable by programs or other libraries as shown in Figure 3.5. Figure 3.5 reflects the view which users have. Details of implementations of such set of functions are not of users' concern and therefore can be safely hidden from the users. The design of the Codeset Conversion API employs the modularity methodology in which system administrators can freely add in their own converters as long as those converters conform to the interface definitions [14].

The APIs are implemented as a library which consists of several conversion modules as shown in Table 3.2, each module converts data from one codeset to another with

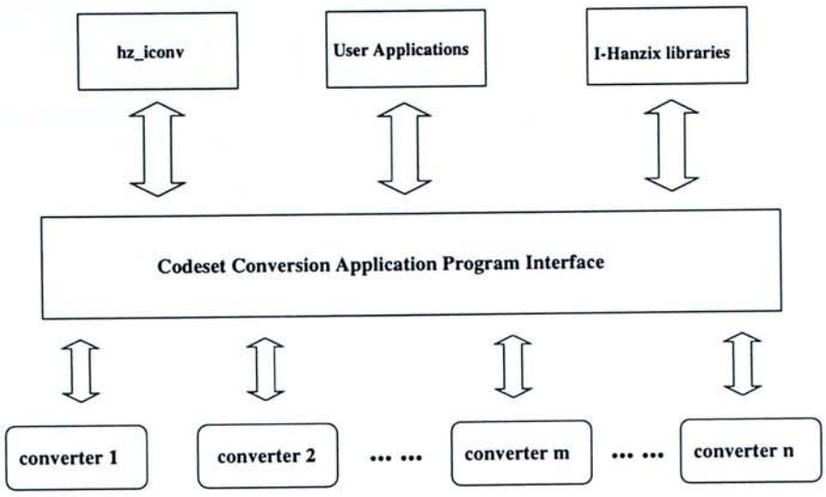

Figure 3.5: The Structure of the Codeset Conversion API

Table 3.2: Different Codeset Converters

Source_Codeset	Target_Codeset	Converter
GB	Big5	GB_BIG5
GB	CNS	GB_CNS
Big5	GB	BIG5_GB
Big5	CNS	BIG5_CNS
CNS	GB	CNS_GB
CNS	Big5	CNS_BIG5

uniform interfaces as seen by the outside world. Due to the bulky size of the library and for the sake of efficiency, user programs are arranged to link with the library dynamically and the library itself is sharable by all programs. All references from the programs to the APIs within the library (libiconv.so, to be precise) are resolved at run-time rather than in the process of compilation.

The conversion API consists of three conversion routines: *hz_iconv_open*, *hz_iconv* and *hz_iconv_close*. The calling sequence of the conversion routines is illustrated in Figure 3.6. Before a converter can be used, the function hz_iconv_open() has to be called, which will attempt to locate the corresponding converter to convert a given source codeset(source) to the given target codeset(target). If there is no converter available to convert source to target, an error status will be returned, otherwise an opaque type pointer will be returned to the caller. The pointer returned shall be used by the subsequent function - hz_iconv() which actually converts data from source to

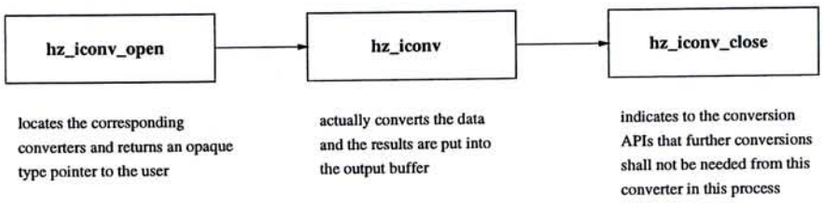

Figure 3.6: Calling Sequence of the Conversion Routines

target. Upon the processing associated with this converter is completed and users no longer wish to use the converter anymore, the function hz_iconv_close() should be called. Here are the definitions of the three routines:

1. `#include <hzconverter.h>`

```
hz_iconv_p   hz_iconv_open(char *target_codeset,
                           char *target_codeset,
                           char *identifier)
```

This function attempts to locate an appropriate converter available in the system for converting *source_codeset* to *target_codeset*. The parameter *identifier* is a modifier which could affect the search of the required converter. Different *identifiers* designate different converters, but they all convert same *source_codeset* to the same *target_codeset*, given that they must be pre-installed into the system and hence being integrated into the codeset conversion library. If *identifier* is NULL, the standard converter for such conversion, if found, will be chosen and returned to the caller. For example,

```
hz_iconv_open("BIG5", "GB", NULL)
```

means a standard converter for conversion from Big5 to GB is opened, and the statement

```
hz_iconv_open("BIG5", "GB", "my")
```

will do the same conversion, but the converter is different which is designated by the third parameter - "my".

2. `#include <hzconverter.h>`

```
size_t  hz_iconv(hz_iconv_p cd,
```

Table 3.3: Exceptional Cases in Codeset Conversions

Categories	Descriptions
1-to-N	A single character in codeset A can be mapped into more than one character in codeset B. This is very common when converting CNS-11643 data into GB-2312, as in other cases where codeset A is a subset of codeset B.
N-to-1	Many characters in codeset A can be mapped into a single character in codeset B. This is very common when converting GB-2312 data into CNS-11643, as in other cases where codeset A is a superset of codeset B.
1-to-0	There is no corresponding character when mapping a character from codeset A to codeset B.

```
char **inbuf,
size_t *inlen,
char **outbuf,
size_t *outlen,
unsigned int *exception_buf,
unsigned int *nchars,
CONVERSION_INFO custom_data)
```

This is the actual conversion routine which does the job of conversions from one codeset to another. hz_iconv() will convert as much data as possible until some exceptions happen.

3. `#include <hzconverter.h>`

   `void hz_iconv_close(hz_iconv_p cd)`

   When a converter is no longer required, users should call this routine so as to free any redundant memory allocated. The parameter cd should be a valid return value from a previous call of hz_iconv_open().

During the conversion, the mappings between characters in one codeset with characters in another codeset is not always a 1-to-1 relationship. Table 3.3 lists the three different cases. In our web system, automatic codeset conversion is carried out through invoking the three conversion routines mentioned above, and linking to the library dynamically. To simplify the conversion and for the sake of efficiency, all exceptional

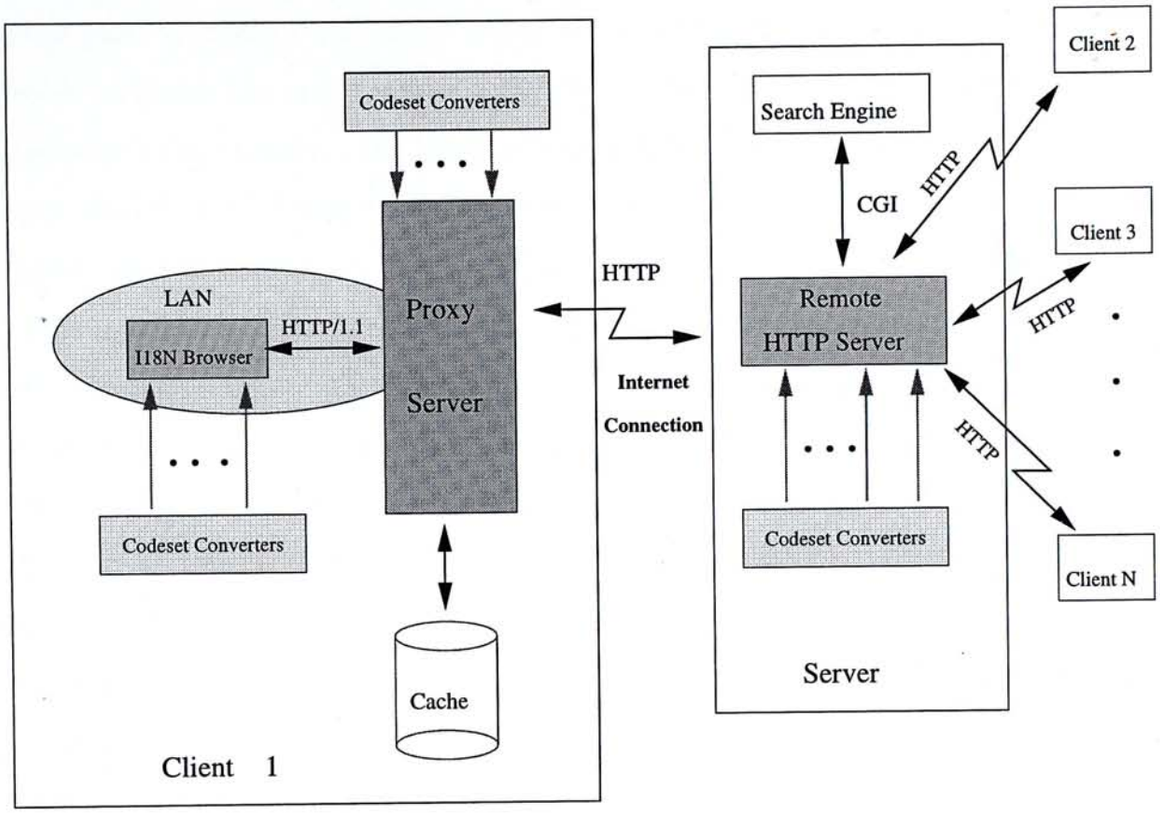

Figure 3.7: Overall System Architecture.

cases for character mapping are handled uniformly, and always use default settings when 1-to-N mapping is encountered.

With the codeset conversion API, it is very easy for our web system to call different converters to do the codeset conversions. And with the uniform definition of API, new converters can be added into the system, such as converters to and from Unicode, in the future.

3.3 Overall System Architecture

3.3.1 Architecture of Our Web System

Through discussions for different cases, we have adopted proxy server into our design for the maximum capability of the system. The overall system architecture is illustrated in Figure 3.7. The whole system is divided into two parts: the client part and the

server part. Although the proxy server is also called a server, it is convenient and flexible to locate the proxy server in the same subnet with clients. When invoking an internationalized browser, the browser is initiated according to the current Locale of user's choice. The codeset converters are installable routines which are based on the Hanzix conversion interface [12, 13, 14] so that they can be plugged into the browser . This internationalized browser is an enhanced browser which supports HTTP/1.1. If the remote server supports pre-HTTP/1.1 only, a proxy server is needed, otherwise the browser can communicate with the server directly. Once the user sends a request with the codeset preference, the browser forwards this request to the remote server or the proxy server. The proxy server checks the codeset information and forwards this request to the remote server. After the HTTP connection, the server tries to find out the retrieved document, either locally or asking helps from other sites through common gateway interface (CGI) [11, 15]. If the original codeset of the retrieved document is different from the preferred codeset, automatic codeset conversion is carried out and the converted document is sent back to the client side. After the proxy server receives the document, it may carry out further codeset conversion if needed, and then maintains a copy of the document in the local cache in case another browser asks for the same document. The proxy server can retrieve a buffered document and return it to the client at a very high speed. If the browser finds that further codeset conversion is needed, it invokes the related codeset converter to carry out another round of conversion. At last, the final document is displayed on the screen without any problem.

In a nutshell, we will provide three packages to users:

1. An enhanced server supporting HTTP/1.1 and having codeset conversion facility.
2. A proxy server with built-in automatic codeset conversion facility and caching mechanism.
3. An internationalized browser supporting HTTP/1.1 and with codeset conversion facility.

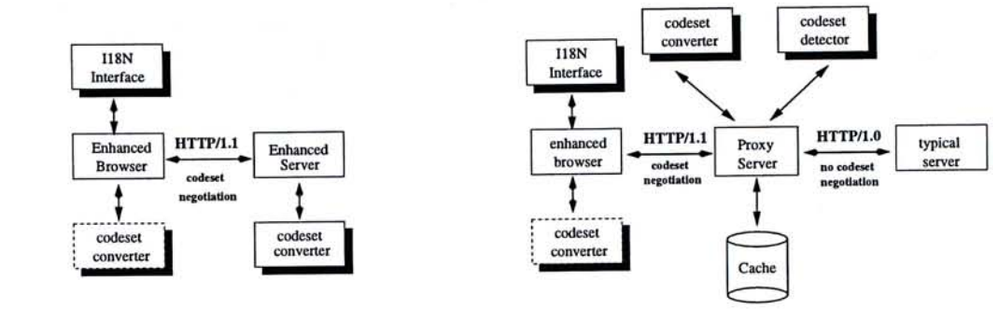

Figure 3.8: Case I and Case II

3.3.2 Flexibility of Our Design

As discussed before, the support of automatic codeset conversion is based on *codeset announcement* mechanism which requires codeset negotiation between the client and the server. This can be carried out through the data type negotiation provided by HTTP/1.1 protocol [9].

However most of the current web browsers and web servers support pre-HTTP/1.1, such as HTTP/1.0 [21]. Besides providing an enhanced server and an enhanced browser supporting HTTP/1.1, the system should be compatible with web systems which support pre-HTTP/1.1 protocol. This brings our design requirements of flexibility for backward compatibility. The component integration approach is used in our architectural design, where each component is independent and reusable, and all components can act in flexible combination to provide services under different situations. There are four cases for the framework of our system which are illustrated in the following figures. All rectangles with shadows are characteristic modules in our web system. Those drawn with dashed lines are optional modules or functions.

Case I: Enhanced Browser Accesses Enhanced Server

Figure 3.8(a). shows that an enhanced browser can communicate with an enhanced server without any problem. The enhanced browser provides an internationalized interface for users, and both the server and browser carry out data type negotiation according to HTTP/1.1 protocol.

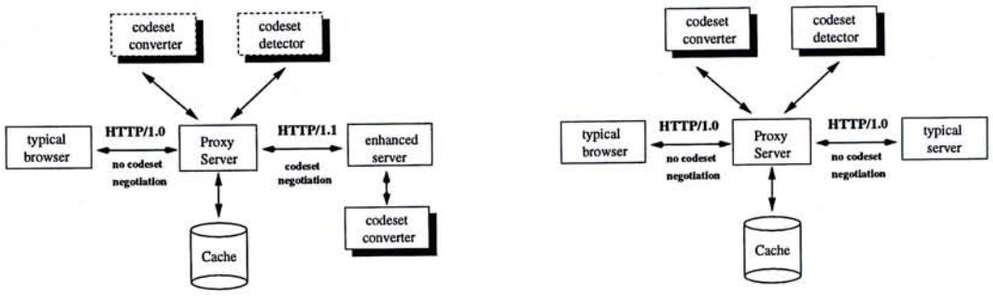

Figure 3.9: Case III and Case IV

Case II: Enhanced Browser Accesses Typical Server

Figure 3.8 (b). illustrates if an enhanced browser wants to access documents on a typical server. Since the typical server will ignore the codeset announcement information sent by the enhanced browser, and the typical server doesn't have any codeset conversion facility built in, an enhanced proxy server must be added as a bridge between them. In this case, the proxy server accepts the request from the enhanced browser, and forwards it to the typical server. After receiving retrieved document from the server, the proxy server will try to identify the codeset of the retrieved document and do automatic codeset conversion if necessary. Then it returns the converted document to the browser.

Case III: Typical Browser Accesses Enhanced Server

Figure 3.9 (a). shows another case where a typical browser wants to access an enhanced server. Although the enhanced server can do codeset conversion, however, the typical browser doesn't send any codeset announcement information to the server, there is no way for the server to know which codeset it should convert the document to. As a result, an enhanced proxy server is also needed in this case. In general, a proxy server is located within a local network. The assumption used by such proxy server is that it regards that most of the web browsers within the local network support an identical codeset. For instance, most web browsers in Hong Kong support Big5, so that the proxy server can announce this codeset to the enhanced server and automatic codeset conversion can be done based on this information.

Case IV: Typical Browser Accesses Typical Server

If users don't have our web software at hands, another framework is also provided for them, which is shown in Figure 3.9 (b). In this case, an enhanced proxy server is a must if users still want to have the specific services. The proxy server assumes that a particular codeset is supported by most web browsers in the local network, and then it accesses the document on the server on behalf of these web browsers. It is indispensable to enhance the proxy server with automatic codeset detection facility so that it can identify the original codeset of the retrieved document by investigating the source code of the document, and carry out codeset conversion if needed. The enhanced proxy server with caching capability also speeds up the document retrieval when a web browser accesses the document which has been accessed by another browser before. The proxy will fetch it from the cache instead of connecting with the remote server again.

3.3.3 Which side do the codeset conversion?

Through discussion in Section 3.3.1, we know a new web system may consist of an enhanced browser, an enhanced server and probably an enhanced proxy server. Consider the case where the browser, the server and the proxy can all do the codeset conversion, an interesting question to be asked is: who is responsible for the actual conversion process? The answer is that it should be more flexible if it can be done nearer to the browser's side, in fact, it is best for the browser to do the job. The reason is that users may sometime provide their own versions of converters, which might be more effective than the default ones as supported by the proxy and/or the server. If we force the users to use the default converters, they will have no choice but accepting the possibly relatively less efficient converters.

3.3.4 Caching

There is a question concerning the caching of data in our model. Assuming that the proxy is to do the codeset conversion on a particular incoming document, there is a dilemma that the proxy may cache the original document or it can cache the converted

form of the document. In principle the proxy may also cache both the unconverted and the converted formats, but this is considered as a waste of space when web documents are usually small in size and therefore it won't cost much in CPU power to convert from the original documents to other codesets. Therefore we have decided that the proxy should cache the original documents only under the control of the installer of the proxy. For many browsers, they have options to allow the retrieved documents to be cached in users' selected directories. Therefore it is decided that it should be users' responsibilities to cache the converted document should one be required. Also it is common for hundreds of users to access one single proxy, it seems to be impractical to cache all the converted documents at the proxy.

Chapter 4

Design Details of An Enhanced Server

As mentioned in Chapter 3, an enhanced web server with automatic codeset conversion functionality should be set up in a new web system to fulfill the additional requirements for Chinese text data processing through Internet. The enhanced web server is built on UNIX platform similar to other English web servers. It has the capability to manage Chinese text data encoded in different codesets on the same server. When data stored in the server are incompatible with what the client machine can process, the server can also provides automatic codeset conversion transparently to client machines.

4.1 Architecture of The Enhanced Server

The architecture of the enhanced web server is illustrated in Figure 4.1. We know, to realize all objectives mentioned in Chapter 3, the enhanced web server must know how to analyze the HTTP request message containing preferred codeset information by the user at the client side, and after the codeset conversion if necessary, compose a new HTTP response message containing the content language and codeset information of the converted/unconverted document so that the browser can parse it and do properly on the documents. All lightly shadowed modules in Figure 4.1 stand for modified

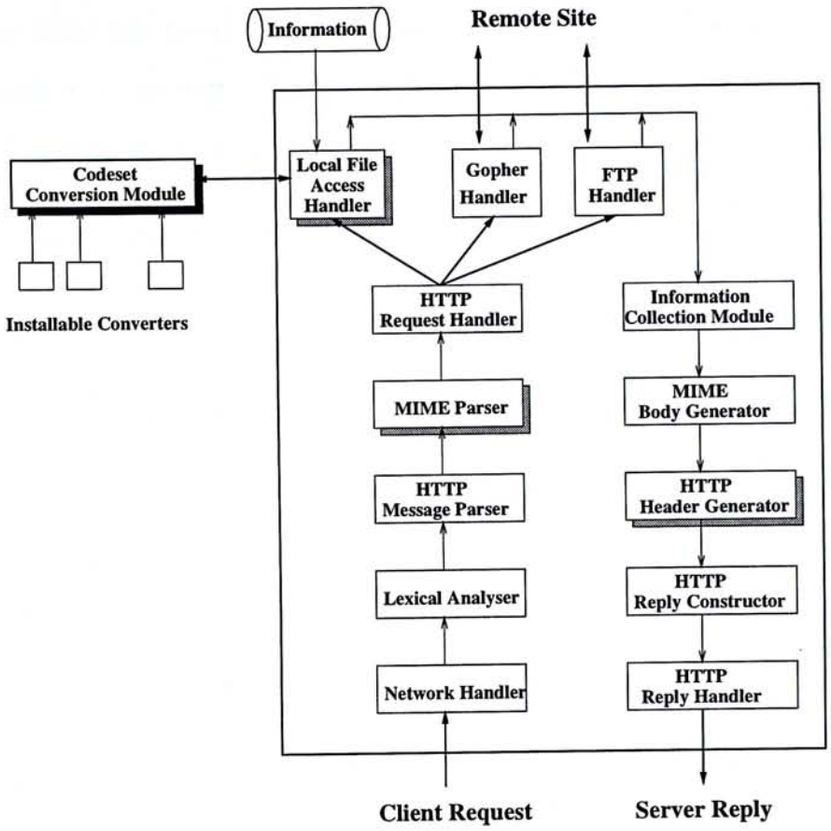

Figure 4.1: The Architecture of The Enhanced Web Server

functions while the heavily shadowed module stands for the new function – codeset conversion.

4.2 Procedure on Processing Client's Request

The steps about how the enhanced web server processes client's requests are listed as follows:

1. **Network Handler**

   Before a web client sends a request to the web server, it first establishes an HTTP connection with the server. Module *Network Handler* is responsible for handling the connection with the client and if there are data coming from the client, *Network Handler* collects them and sends them to the next module.

2. **Lexical Analyzer**

Like any language parser, module *Lexical Analyzer* is designed to do the lexical investigation of the input data. And it tries to detect any syntactical errors of the data and pass the correct part of the data to the next processing module.

3. **HTTP Message Parser**

   In general, an HTTP message consists of three parts: a header-line, message header fields, and the message body. Module *HTTP Message Parser* divides the three parts into different objects and invokes module *MIME Parser* for further parsing.

4. **MIME Parser**

   MIME Parser is an important part of the web server. It is responsible for the analysis of the HTTP request message's header fields. It has been modified to fulfill the additional requirements, i.e. it is capable to analyze the codeset announcement information in the HTTP request message which is sent by the web client. This will be described in more details in the next section.

5. **HTTP Request Handler**

   In the header-line of each HTTP request message, there is a part describing what the protocol the web client uses, and module *HTTP Request Handler* is right for identifying this information and invokes different protocol handlers to do the further process.

6. **Local File Access Handler**

   If the web client wants to access a document located on the local server other than remote servers, this module tries to locate the document from the file system of the server. *Local File Access Handler* module has been enhanced to have the capability to detect the codeset of the original document which will be explained later. If the codeset of the document is incompatible with what client requests, codeset conversion should be carried out later.

7. **Codeset Conversion Module**

This is a new module which is added into the server to do the automatic codeset conversion. It is the interface between the web server and the Hanzix codeset conversion libraries and routines which are described in Section 3.2.3. This module tries its best to convert the retrieved document on the same server from the original codeset to the one required by the web client.

8. **Gopher Handler, FTP Handler**

   These two modules are similar to module *Local File Access Handler* and are designed to handle requests for Gopher and FTP protocols respectively. Both of them need to access the remote site to get back the data required by the web client.

9. **Information Collection Module**

   After *Local File Access Handler*, *Gopher Handler* or *FTP Handler* gets the data from related server, no matter codeset conversion has been carried out or not, all data will be collected by module *Information Collection Module* for further processing in the next step.

10. **MIME Body Generator**

    This module composes the HTTP response message's body part with the input data from the *Information Collection Module* and rearrange them into the format according to HTTP specification.

11. **HTTP Header Generator**

    This module is responsible for generating the header fields of each HTTP response message. It has been enhanced to notify the web client what codeset of the data in the body part is. The codeset notification is done through a header field *Content-type*'s charset parameter.

12. **HTTP Reply Constructor**

    This module composes the complete HTTP response message through combining both the message body generated by *MIME Body Generator*, and the message header fields generated by *HTTP Header Generator*.

13. **HTTP Reply Handler**

    This module uses the HTTP connection established at the very beginning, and sends the HTTP response message to the web client through network.

4.3 Modifications of The Enhanced Server

To enhance the server to be able to carry out data type negotiation, and automatic codeset conversions, four kinds of modifications have been completed: the interpretation of the client's codeset announcement information, codeset identification of the retrieved document on the local server, codeset notification to the web client and automatic codeset conversion.

4.3.1 Interpretation of Client's Codeset Announcement

According to the HTTP/1.1 standard definition, there is a field "Accept-Charset" which can let the web client to inform the server about the client's preferred codeset. There is also another field called "Accept-Language" which is used by the client to notify the server what language it can accept. However, these two fields are not supported by the current implementation in most of the business web browsers, such as Netscape and Internet Explorer. Also, most of the business web servers, say, NCSA and CERN's httpd now only support HTTP/1.0 specification, therefore they also ignore these two fields.

Under the implementation of our enhanced web browser, we can notify the codeset and the language information inside the HTTP request header. Figure 4.2 shows a simple HTTP request message sent out by our enhanced browser. It is noticed that the web client can accept codeset "gb2312 (simplified Chinese)", and language zh_CN (Chinese), en (English). And the protocol the client uses is HTTP/1.0 (Although we support the new features of HTTP/1.1, we do not support HTTP/1.1 completely, thus we still use HTTP/1.0 when requesting service from our server).

GET /Index.html HTTP/1.0
Date: Sun, 20 Apr 1997 19:00 GMT
Accept: text/plain,text/html,image/gif,*/*
Accept-Charset: gb2312
Accept-Language: zh_CN, en
Host: cweb1.cs.cuhk.hk:8888
User-Agent: NCSA_Mosaic/2.6 (X11;SunOS 5.5 sun4u)

Figure 4.2: HTTP/1.1 Request Message From Our Enhanced Browser

Our enhanced server should be modified to extract the codeset information from HTTP request header field "Accept-Charset". The first thing to do is to add this codeset field into the *HTRequest* data structure. *HTRequest* is an object which stores the HTTP request information from the client. The line added in *HTRequest* is shown below:

```
HTList* charsets;  /* accepted charset */
```

HTParseRequest() is a function which gets an HTTP request message from the network channel. It parses the request message and then puts it into *HTRequest* data structure. A new defined function *HTAcceptCharset()* has been added into *HTParseRequest()*. Its main function is to get the codeset information and add it into data structure *HTRequest*. After executing the new function, all codesets in "Accept-Charset" from the HTTP request header will be stored into another link list for later use. All these modifications are carried out in module *MIME Parser* shown in Figure 4.1.

4.3.2 Codeset Identification of Web Documents on the Server

After the server knows what the preferred codeset required by the web client, it has to fetch the file in the system and tries to identify the original codeset of the retrieved document to carry out codeset conversion if the two codesets are incompatible.

At present, we have the following ways (in descending order) when determining the codeset of a given web document as perceived by our web server:

```
(a) by scanning the special HTML tag - namely "<LANG>" within
```

```
    the requested document.
    e.g. "<LANG = gb2312>" denotes the following content is
         written in simplified Chinese with codeset gb2312.
(b) by querying the I-Hanzix server which records the codeset
    of virtually every file in the system.
(c) by inspecting the file extensions, e.g. files having the
    extension htmlgb will be taken as GB files, and file
    extension htmlb5 will be taken as Big5 files.
```

The server first investigates the HTML document to see whether there are <LANG> tags. If there are, the server extracts these information first, and then tries to convert all data encoded in different codesets into data encoded in the same target codeset if possible. Here is an example of multi-codeset HTML document:

```
<HTML>
<BODY>
<LANG = gb2312>
......
<LANG = big5>
......
</LANG>
......
</LANG>
......
</BODY>
</HTML>
```

The above data including <LANG> tags are regarded as 3 blocks.

```
The first block  -- between <LANG = gb2312> and <LANG = big5>,
The second block -- between <LANG = big5> and the first </LANG>,
The third block  -- between the first </LANG> and the second </LANG>.
```

The first block is considered written in simplified Chinese with codeset gb2312, the second block is regarded written in traditional Chinese with codeset big5 and the third

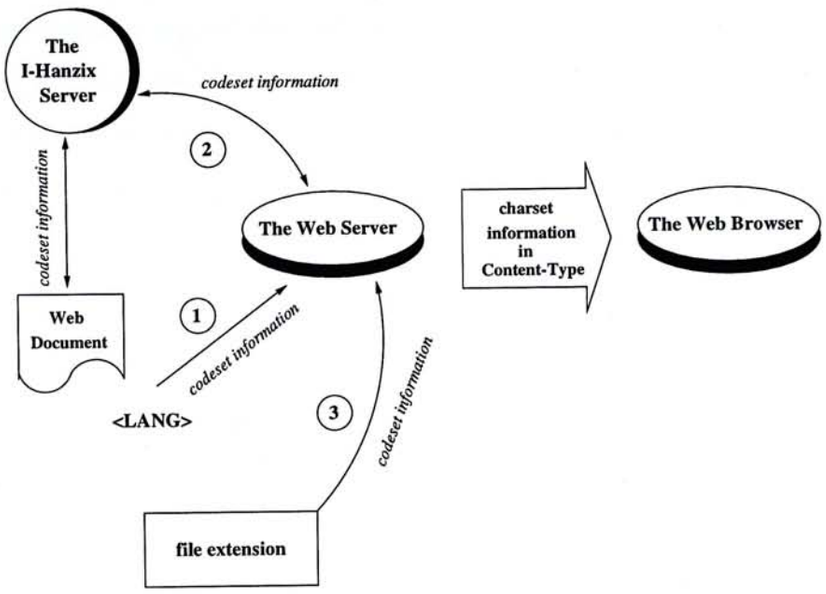

Figure 4.3: Steps in Determining the Codeset of a Given File

block is considered written in simplified Chinese again according to the embedded structure of <LANG> tags. Suppose, the codeset preferred by client users is gb2312, then the first and third blocks don't need to be converted, only the second block needs to be converted from big5 to gb2312. Then the converted document will be sent out for further process.

If the server finds there are no <LANG> tags inside the retrieved document, then it relies on the I-Hanzix server [13, 14] to obtain the codeset information for the corresponding file. Therefore it will try to connect to the I-Hanzix server, if it is available in the system.

In case both the (a) and (b) failed, the server can only rely on the file extension of the requested file to determine the file codeset. As discussed before, the codeset of document with file extension *htmlgb* is regarded as GB while the codeset of document with file extension *htmlb5* is regarded as Big5. Figure 4.3 shows the steps in determining the codeset of a given file by our server. A new function called *codeset_ident()* is added into our server. And it invokes three other new-developed functions: *get_codeset_by_extension()*, *get_codeset_by_IHanzix()* and *get_codeset_by_HTML()* to realize the codeset identification by three methods. Their definitions are as follows:

```
char *codeset_ident(char *Full_Path_Filename);
```

```
char *get_codeset_by_extension(char *Full_Path_Filename);
char *get_codeset_by_IHanzix(char *Full_Path_Filename);
char *get_codeset_by_HTML(char *Full_Path_Filename);
```

Each method determining the codeset of the retrieved document has its own advantages and disadvantages. For example, the last method relying on file extension seems to be the simplest one, however, since the limitation of the "*DOS*" which has only 3 characters for file extension, this method is not suitable when the server is running on a PC. The obvious drawback of the second method is that it relies on the I-Hanzix server which has not yet been very popular. Regarding to the first method, let the server to parse the html file itself and find out the codeset information will decrease the performance of the server.

In the future, automatic codeset detection will be added into our web system. And the server relies on the detection module to identify the codeset of the retrieved document. Although this will also decrease the performance of the server, but it is considered to be the least dependent approach comparing with other methods. Because, the file extension convention may not be accepted by other users, also the <LANG> tags may also not be used by most of the HTML authors, and it is not practical to force users to install our I-Hanzix server if they don't want to.

4.3.3 Codeset Notification to the Web Client

Codeset notification involves putting the codeset information of the returned document into the HTTP response message. In HTTP/1.1 specification, one of the attributes of the Content-Type header field – charset can be used for this purpose. There are two cases when the server deals with the codeset notification to the web client, one with codeset conversion and the other without codeset conversion.

When codeset conversion is involved, the "charset" attribute should be set to the target codeset of the conversion. This codeset information is the result of interpreting the web client's codeset announcement information. On the other hand, without

codeset conversion, the "charset" attribute should be set to the original codeset of the retrieved document. This codeset information can be obtained by the codeset identification step.

Function *HTReplyHeaders()* is responsible for generating an HTTP response message's header according to the corresponding HTTP request by the client. It has been modified to add codeset information into the charset attribute of the header field Content-Type. The following examples show the HTTP response header generated by the server for both cases:

```
Case I:
   HTTP/1.0 200 OK
   Date: Sun, 25 May 1997 15:23:52 GMT
   Content-Length: 2476
   Content-Type: text/html;charset=gb
   Expires: Tue, 27 May 1997 15:23:52 GMT
   Last-Modified: Mon, 19 May 1997 7:17:58 GMT
```

In this example, no codeset conversion is involved, so the server returns the original codeset information - "gb" to the web client.

```
Case II:
   HTTP/1.0 200 OK
   Date: Sun, 25 May 1997 15:23:52 GMT
   Content-Length: 2476
   Content-Type: text/html;charset=big5
   Expires: Tue, 27 May 1997 15:23:52 GMT
   Last-Modified: Mon, 19 May 1997 7:17:58 GMT
```

In this example, codeset conversion from *gb* to *big5* has been carried out, and the target codeset *big5* is returned to the web client.

4.3.4 Codeset Conversion

A function *file_convert()* is designed as the interface between the server modules and the Hanzix's codeset conversion library [14]. The prototype of this function is shown below:

```
int file_convert(in_fptr, out_fptr, from_codeset, to_codeset)
FILE *in_fptr, *out_fptr;
char *from_codeset, *to_codeset;
```

The *in_fptr* is a pointer points to the file which the client requests. The *out_fptr* is a temporary file which stores the converted version of the requested file. The *from_codeset* is the codeset of the original file which is the result obtained through the codeset identification by the server. The *to_codeset* is the target codeset obtained by the interpretation of client's codeset announcement.

The reason for storing the converted data into a file is that simple caching mechanism may be applied for improving the server's performance. When clients request the same data next time, codeset conversion would not carry out twice.

To ensure the safety, the output buffer is much larger than the input buffer to reduce the chance of overflow. It is potential that the converted data occupy much larger space, for example, CNS character may occupy 4-bytes for each character while GB or Big5 only use two-bytes per character, so when converting a GB character into a CNS one, the latter must have enough space to hold the converted data.

4.4 Experiment Results

Figure 4.4 shows how the enhanced server realizes the codeset identification of the document written with <LANG> tags, and carries out the codeset conversion. The right window shows that the current font setting is gb2312, and the middle window shows that the original codeset of the retrieved document "funny1.html" is Big5. The left window shows that the converted document encoded in gb2312 is displayed on the

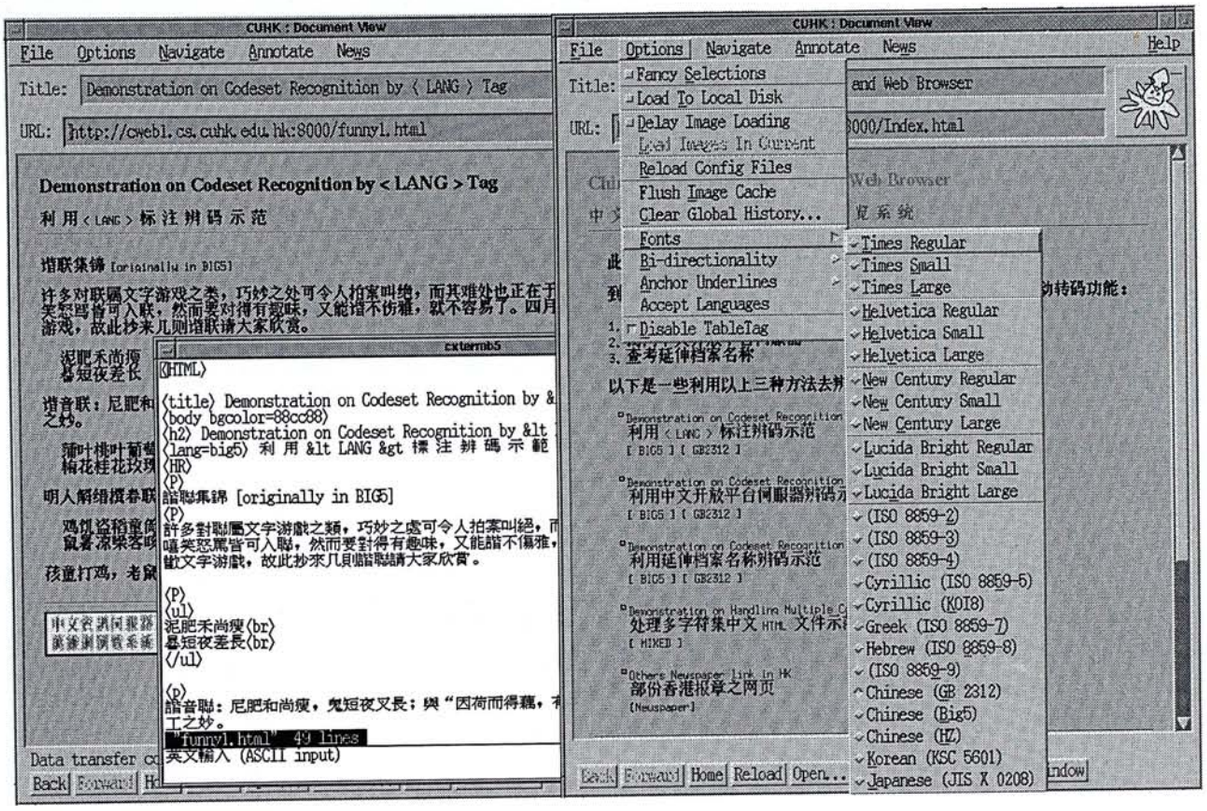

Figure 4.4: Demonstration on <LANG> Tag Identification By The Enhanced Server

screen. In this example, codeset conversion from Big5 to gb2312 has been conducted by the server after it analyzes the <LANG> tags in the retrieved document.

Figure 4.5 illustrates that the server can identify the codeset of the retrieved document by its name's extension. Since the font setting for the browser is Big5, the server does the codeset conversion from gb2312 (the original codeset of the document "vr.htmlgb") to Big5.

Figure 4.6 shows another example. The right window is the result of accessing the mixed.html file using Netscape. File mixed.html is written in multiple codesets: gb2312, Big5 and CNS. And the multi-codeset document is written using <LANG> tags which is shown in the middle window. Since Netscape doesn't have <LANG> tag analysis and doesn't have codeset conversion facility built in, users set the "Document Encoding" to simplified Chinese (gb2312), only the line written in gb2312 can be shown correctly, other lines are all garbage data. Using our web system, the browser informs the server that its preferred codeset is gb2312, and the enhanced server analyzes the

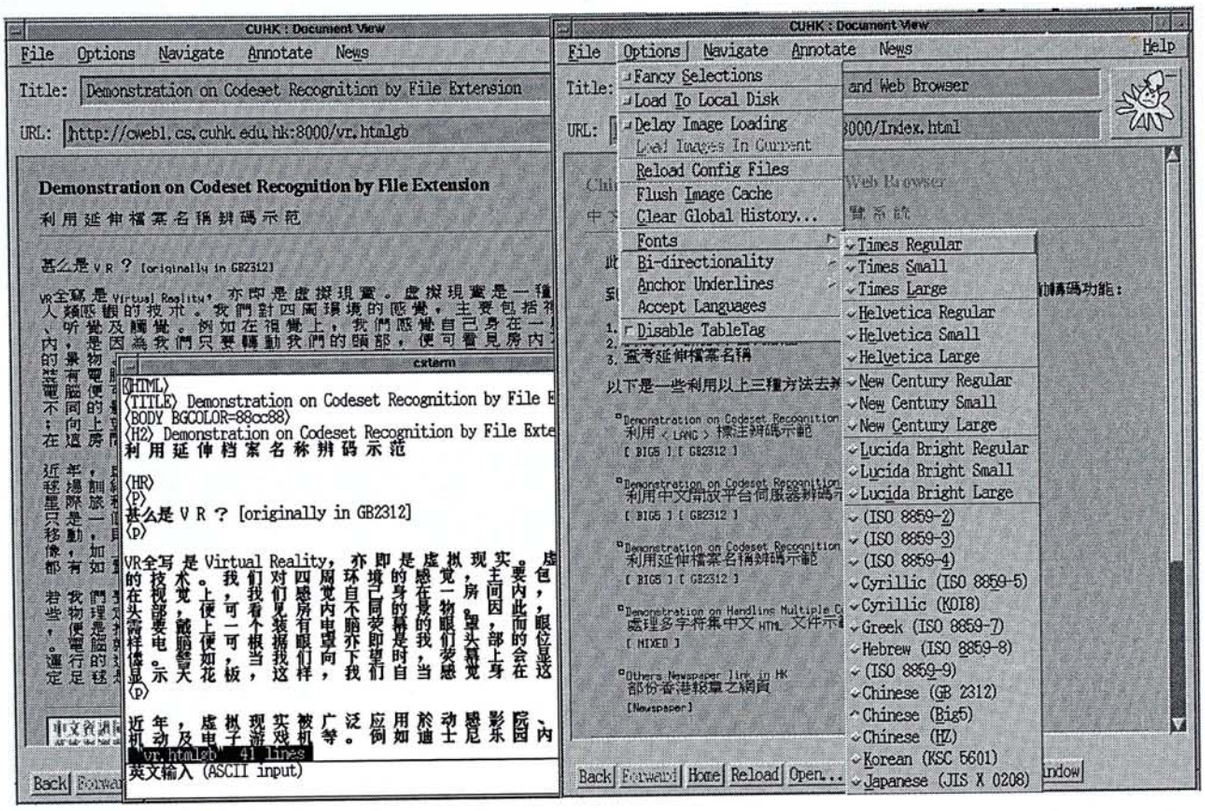

Figure 4.5: Demonstration on Codeset Identification By File Extension

multi-codeset document and converts all lines written in codesets other than gb2312 to the same target codeset - gb2312, at last, the converted document is sent back to the browser.

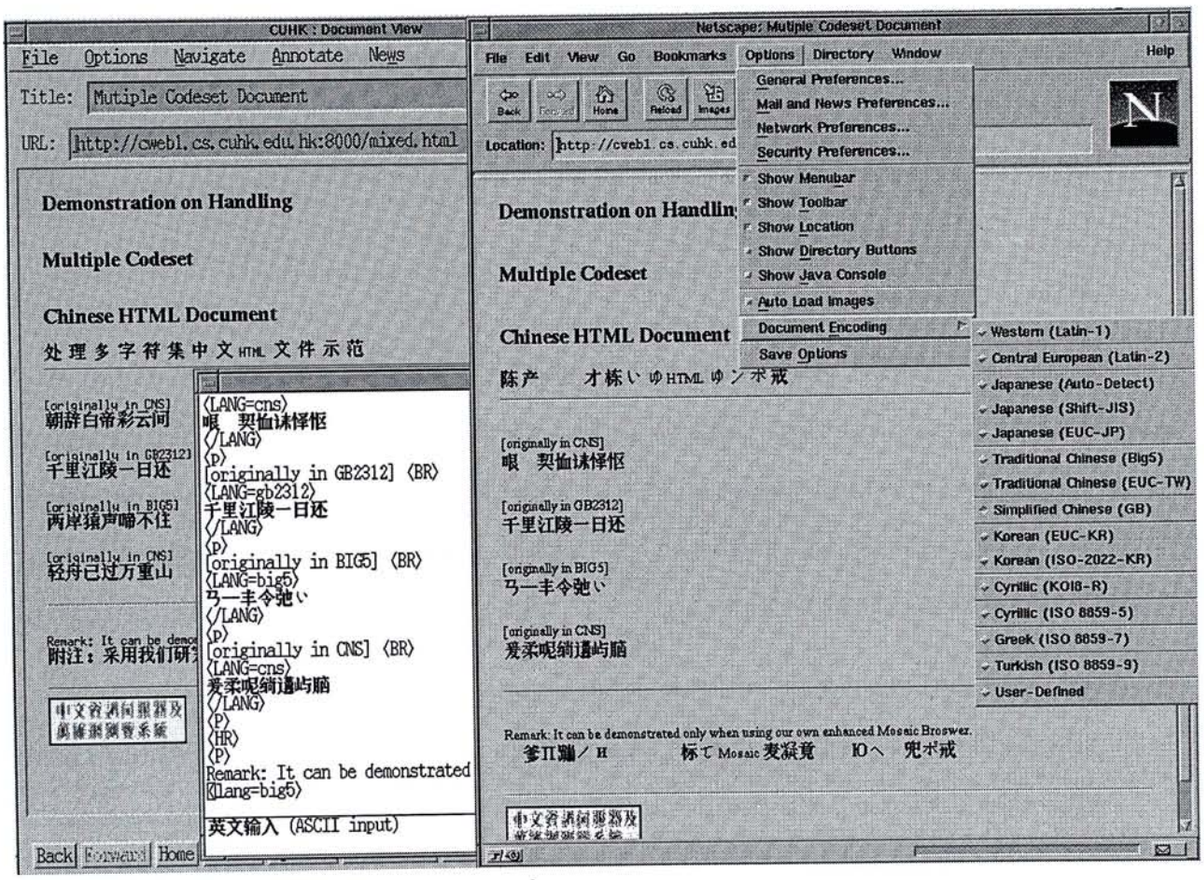

Figure 4.6: Demonstration on Handling Multi-Codeset Chinese Document

Chapter 5

Design Details of An Enhanced Browser

The enhanced web browser is developed based on Mosaic which is a free software in the public domain. Like any other web browsers, the main function of Mosaic is trying to establish the HTTP connection with the remote server which users want to access, sending the request to the server, and handling the response from the server and interpreting the returned document and displaying it on the browser interface.

The enhanced browser is running on UNIX platform and it can handle Chinese text data encoded in different codesets, such as GB, Big5 and CNS. It also provides codeset conversion if required by users. The browser interface has been internationalized. It allows users to access server information either using traditional Chinese or simplified Chinese without the need to match the server's codeset.

5.1 Architecture of The Enhanced Browser

Figure 5.1 shows the architecture of the enhanced browser. In this figure, all lightly shadowed rectangles stand for modified functions while the heavily shadowed rectangle stands for the new feature added into the system.

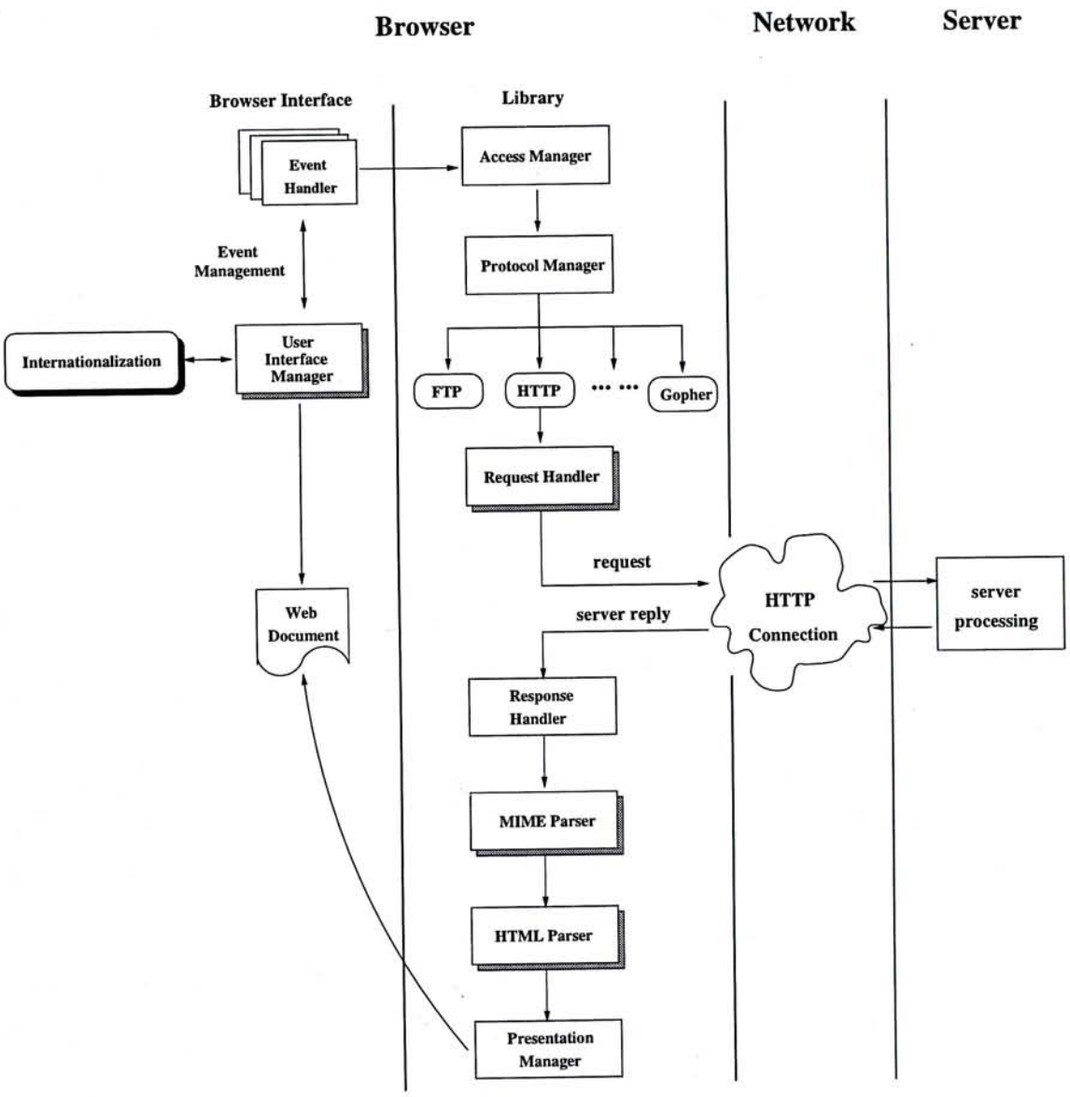

Figure 5.1: Architecture of The Enhanced Browser

The browser system consists of two main parts: *interface* and *library*. The *library* used in the browser system is *libwww version 2.0*, and it is a general code as the basis for building the browser. The *library* contains code for accessing servers with various protocols, such as HTTP, FTP, Gopher and etc. In addition, it provides functionality for loading, parsing and caching graphic objects plus a wide spectrum of generic programming utilities [29]. The *interface* part is responsible for setting up browser windows with menubars, buttons, document view windows and etc. The *interface* part is developed on top of Motif which is an event-driven programming environment. It is designed to respond to user events triggered by moving and clicking the mouse on the menubar options, buttons or hyperlinks in the web documents. The handling for each event is dealt by corresponding *Event Handler*. Once an event is triggered, the *Event Handler* either communicates with the *User Interface Manager* or invokes related modules in the *library* depend upon the property of the event. In general, if an event just needs the *interface* part's involvement, the *Event Handler* only connects with the *User Interface Manager* for further service, otherwise, the *Event Handler* will invoke the *library* functions to complete the task.

In order to provide a convenient and user-friendly access environment to Chinese users, *Internationalization* [22, 31] feature has been added into the *interface* part of the browser. Every time the browser is invoked, it may set its whole environment automatically by checking the *locale* [22, 31] on the local machine, or users can manually change system configuration and environment through setting UNIX environment variables. Therefore, users can choose to set the browser interface in either traditional Chinese or simplified Chinese, such that all menubars, buttons and other prompt messages are all in Chinese.

Furthermore, the browser must be enhanced to carry out data type negotiation with web servers. The enhancements have been done in the *library* part. The browser is capable to announce the preferred codeset to the server when it sends out each HTTP request, and after it receives the server reply, it knows how to interpret the additional codeset information contained in the HTTP response message if there are, and carries out automatic codeset conversion if the codeset is incompatible with what users prefer.

Also the browser has the capability to analyze the <LANG> tag of HTML/3.0 to get the codeset information.

It should be noticed that the interface environment is separated from the document display environment. That is the interface environment will not change once it has been set up at the very beginning. However, the document display format can be changed through selecting different font options in the menubar. For example, a user can choose the user interface to be Chinese, but the browser still can display web documents written in other languages. This scheme provides users with the most flexibility. Suppose a user doesn't know English very well, s/he can choose Chinese interface. In such a case, it is easy for him/her to operate the browser, meanwhile, there is no hindrance for him/her to retrieve English web documents or documents written in other codesets.

5.2 Procedure on Processing Users' Requests

The steps about how the enhanced browser processes users' requests are explained blow:

1. **User Interface Manager**

   This module deals with all interface management, such as establishing a new window, menubars, buttons, icons, loading the retrieved document, generating dialog boxes, changing the fonts of the document, etc.. All the handling of menubars, buttons, dialog boxes has been modified to realize the internationalization. And some codes have been added into this module to carry out codeset conversion if required by users.

2. **Event Handler**

   There are a lot of event handlers corresponding to the many events. Once an event is triggered, the corresponding event handler will be invoked to handle it. *Event Handler* either calls functions of the *interface* part, or invokes functions in the *library* part to complete the task.

3. **Access Manager**

Once an event related to server access is triggered, this module will be invoked. Its main function is to analyze the URL information sent to it, and save the current window document into the cache and remember the trace with the help of history list etc, then it gives the control to the protocol module to handle various kinds of server access.

4. **Protocol Manager**

   The *Protocol Manager* extracts the protocol information from the URL first, then establishes the connection with the remote server (or the local file system) using related protocol obtained by *Access Manager.*

5. **Request Handler**

   This module composes the HTTP request message and sends it to the remote server via the HTTP connection. It has been enhanced to post the preferred codeset information in the request header so that the codeset announcement can be realized by the client side.

6. **Response Handler**

   This module is responsible for collecting incoming data from the network channel, that is the server reply will reach *Response Handler.* Then it passes all data to *MIME Parser* for further analyses.

7. **MIME Parser**

   This is the module which has been enhanced to be capable to analyze the codeset notification sent by the server in the HTTP response message. It will parse all header fields, and put the information into related data structures. If necessary, codeset conversion will be done in this module.

8. **HTML Parser**

   This module is designed to parse the retrieved HTML document contained in the body part of the HTTP response message, and divides the stream data into different objects and save them into related data structures. It is expected to have the capability to analyze the <LANG> tags in the HTML document if there are, and carry out codeset conversion if needed.

9. **Presentation Manager**

   This module gets the different objects generated by module *HTML Parser* and tries to display the document on the screen.

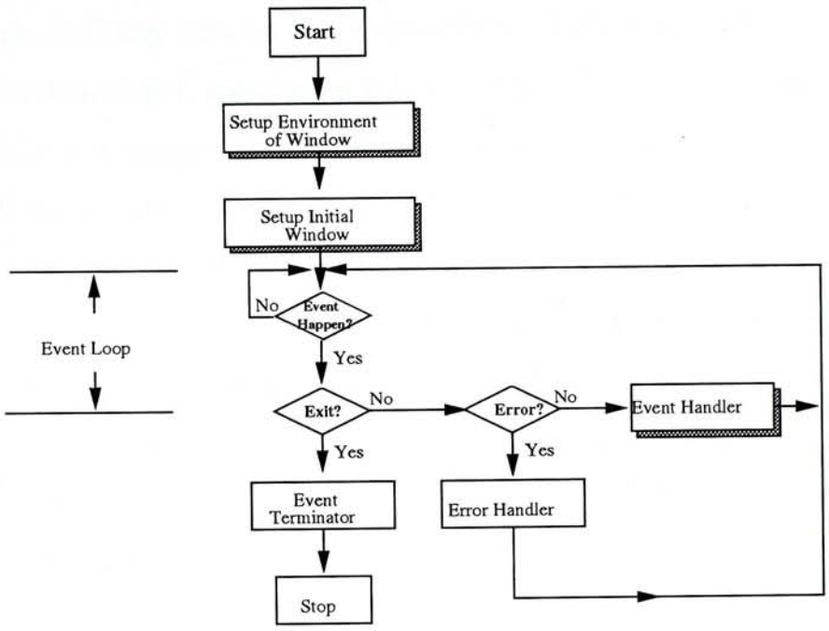

Figure 5.2: Basic Control Flow of Mosaic.

5.3 Event Management and Handling

As discussed in former sections, it is known that there are a lot of events which may be triggered by users. The user interface of the browser is designed to respond to users' event triggering by moving and clicking on the menubar options, buttons or hyperlinks in the web documents. And every event is mapped to a call-back function, referred to as *event handler*. The main program of the browser can be seen as an event loop. When an event is triggered, the browser calls the related event handler to do the corresponding processing, when the handling for this event is finished, the browser goes back to its original state waiting for another event to happen, like a loop. Such situation will not end until a terminator event is triggered, which means the browser will go to its end at last. In the following part, I talk the basic control flow of the browser first, then go through the details of some typical event handlers to see what we have done to fulfill the additional requirements for Chinese processing.

5.3.1 Basic Control Flow of the Browser

The basic control flow of the browser is illustrated in Figure 5.2. When the browser

starts to work, it firstly sets up the environment of window, such as sets up global history list, bookmark list, annotation list, etc. Then it creates the initial window and loads the default homepage document into it. From now on, it is users' round to trigger events to ask the browser to offer services. There are various events, such as to move from this window to next window by clicking button 'back', 'forward' and 'home' etc; to reload the current document; to save, edit or print current window's document, to connect to another site by clicking 'Open URL' or clicking hyperlinks in the document, etc. Each event is handled by related event handler. If an error occurred, an error handler takes over the job and does proper processing. After an event finishes, the browser starts to wait for the next event again until an exit event is triggered.

For an enhanced browser, module `setup environment of window` should be modified to detect and set up window environment according to the *locale* of the local machine. As a result, a localized user interface is established for users. And the initial window must be customized in the same way. There are no added events in the enhanced browser system. Event handlers are needed to be modified/enhanced if the related events involve remote server access, such as event `Open URL` with remote url address, event `Reload`, and event `Clicking Hyperlinks In The Document` which causes remote document retrieval, etc.., or if users reset the font from the menubar, the related event handler has to be modified to carry out automatic codeset conversion.

5.3.2 Event Handlers

To fulfill the project's requirements, operations of event handlers involving remote server access should have additional processing including HTTP message header handling which involves the client's codeset announcement and the interpretation of server's codeset notification, codeset conversion and HTML language tag parsing [18]. Furthermore, if the remote server is not capable to convert the retrieved document into the codeset which is preferred by the client users, the browser should be enhanced to do the on-line codeset conversion by itself. In the following part, I explain two typical event handlers to illustrate what we have done to make the new features realized in

our browser system. The first one is categorized to be one of those events which need remote server access which mostly gets the browser's *library* part involved. And the second one is just related to the browser *interface* part.

Event Handler: Open URL

Among those events which need server access, event `Open URL` needs all steps of operations and is the typical case to show how we add our new functions to enhance the browser. Figure 5.3. shows the control flow of event `Open URL`. In this figure, all shadowed parts are new function modules.

When `Open URL` event is triggered, the `URL handler` generates a dialog box asking users to input URL address, after that `Fetch Text Handler` analyzes the URL site information and saves current window information into buffers. Then, the `Access Manager` distinguishes different URLs by analyzing the protocol part of URL, and then invokes different protocol managers to do proper operations. The `HTTP Protocol Manager` calls `HTTP Connection` to connect the remote server, and then `Request Handler` sends requests containing codeset announcement information, after the server sends back response, the `Response Handler` collects all response message and sends them to the `MIME Parser` to accept parsing. `MIME Parser` must extract codeset information first and check whether it is the same as what users prefer, if not, automatic codeset conversion must be done at `Codeset Conversion Module`, and the converted document is sent to `HTML Parser` for display. If no codeset information is included in the server's response header, that is `MIME Parser` extracts no codeset information, `HTML Parser` should parse the document to check tag <LANG> in the HTML document and do codeset conversion if necessary. Then the document is sent to the `Presentation Manager` for final display.

Modification 1: Codeset Announcement

In Request Handler, codeset announcement should be added so that the local environment of the client can be sent out to the remote server. According to HTTP/1.1, there

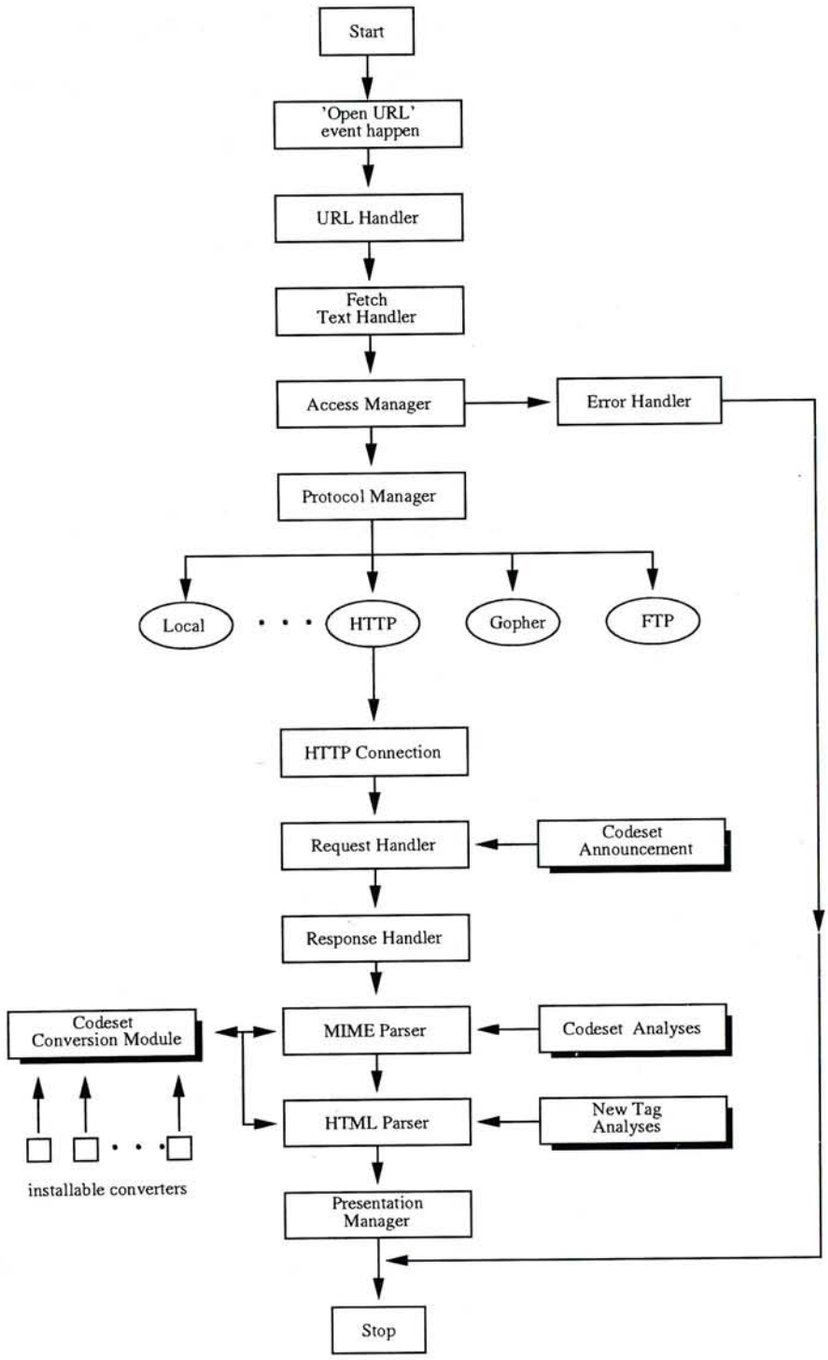

Figure 5.3: Control Flow of Event "Open URL."

is a field "Accept-Charset" which can let the web client to inform the server about the client's preferred codeset. Therefore, a new parameter 'accept_charsets' should be added into the structure mo_window. *mo_window* is an important data structure which keeps all information related to a single "Document View Window", including subwindow details, menubar, button, mail function, history list and so on. The line added into *mo_window* has the following format:

```
char accept_charsets[30];
```

It can be cited as 'win->accept_charsets' where win is a pointer of type mo_window [18]. The definition of variable win is as follows:

```
mo_window *win;
```

The value of accept_charsets can be acquired either through the default *locale* setting at the beginning of invoking client system, or by users resetting the font option in the menubar. Function *mo_set_fonts()* has been modified to fill the accept_charsets parameter if users reset the font. The definition of *mo_set_fonts()* is shown below:

```
mo_status mo_set_fonts (mo_window *win, int size)
```

The size parameter in *mo_set_fonts()* stands for the numeric representation of each font in the menubar. Once users click a certain font, the numeric value of this font will be sent to this function. When variable 'win->accept_charsets' has been filled up, a new function:

```
char *mo_get_accept_charsets()
```

is called to get the right value. Module *HTLoadHTTP()* has been modified to call *mo_get_accept_charsets()* function when composing the HTTP request message.

Modification 2: MIME Parser

After the client sends out the request, the server will send back a reply later. When

the client receives the HTTP response message, it is the MIME Parser who is responsible for the parsing of all header fields, including the interpretation of the codeset notification in the response header. According to HTTP/1.1, the parameter *charset* of response header field *Content-Type* can be used by the server to indicate the codeset information of the retrieved document. Therefore, MIME Parser should be modified to interpret this charset parameter.

The MIME Parser is a Finite State Machine(FSM) parser. And its code is conformed to the specification in RFC 1341. The processing of MIME parsing is based on different states at different time. This MIME Parser is tolerant to all syntax errors. It ignores field names it does not understand, and resynchronizes on line beginnings [19].

There are fifteen states in the MIME Parser, they are shown as follows:

1. BEGINNING_OF_LINE
2. CONTENT_
3. CONTENT_T
4. CONTENT_TRANSFER_ENCODING
5. CONTENT_TYPE
6. CONTENT_ENCODING
7. CONTENT_LENGTH
8. LOCATION
9. SKIP_GET_VALUE
10. GET_VALUE
11. JUNK_LINE
12. NEWLINE
13. CHECK
14. MIME_TRANSPARENT
15. MIME_IGNORE

The relationships among different states are illustrated in Figure 5.4. And here are the explanation of the legend in the diagram:

1. All rectangular boxes are handlement parts. And the color-filled ones are modi-

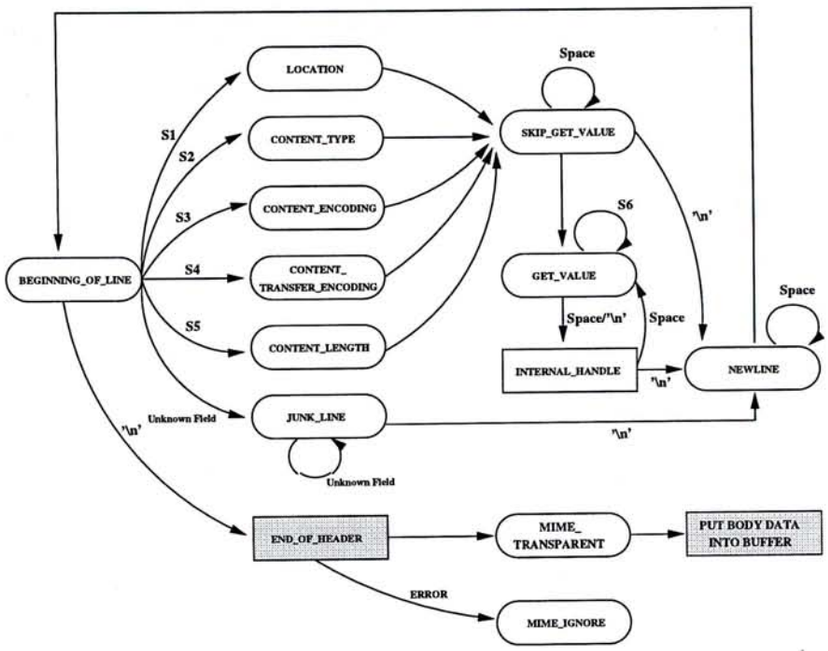

Figure 5.4: FSM MIME Parser.

fied ones.

2. All non-rectangular boxes are states.
3. An arrow means the transition from one state to the other.
4. The characters above each arrow indicate the input string.
5. A cycle beside a state means after a certain character or character sequence is inputted, the state will not be changed.
6. Meanings of all symbols and character sequences used in the diagram:

```
S1 = "Location"
S2 = "Content-type"
S3 = "Content-encoding"
S4 = "Content-transfer-encoding"
S5 = "Content-length"
S6 = The value of a certain HTTP header field, usually a string
Error = fatal error which is unexpected by the MIME Parser.
Unknown field = This field is absolutely unknown by the parser.
Space = '\r' or space character
```

The input stream pushed into the FSM is assumed to be stripped on 'CR's, i.e.

lines end with LF, not CR LF. Each time, when a character is read into the parser, according to the current state, the parser decides what is the next state.

The first state of the whole parsing should be `BEGINNING_OF_LINE`, and the last state is either `MIME_IGNORE` or `MIME_TRANSPARENT`. If an unexpected error occurred during the parsing, the state will be changed to `MIME_IGNORE`. In fact, this state should never happen only in the case that a fatal error occurs. The HTTP header (or the MIME header) fields which can be parsed by this parser are: `Content-Type`, `Content-Encoding`, `Content-Transfer-Encoding`, `Content-Length` and `Location`, states 4, 5, 6, 7, 8 are just corresponding to these fields.

If a line can not be understood by the parser, it simply skips the rest of unknown characters and jumps directly to state `JUNK_LINE`, and when the character '\n ' is met, the state will be changed to `NEW_LINE`. And `NEW_LINE` will change itself to `BEGINNING_OF_LINE`. States 2, 3 are intermediate states before knowing the final field name, and state `CHECK` is the common part for checking all the fields' names with the help of a check buffer keeping all needed matching strings. For simplicity and clarity, Figure 5.4. doesn't include two intermediate states and the `CHECK` state.

The rectangular boxes stand for handlement parts after a certain state.There are three main handlements parts:

`INTERNAL_HANDLE`, `END_OF_HEADER` and `PUT_BODY_DATA_INTO_BUFFER`.

When state `GET_VALUE` meets character '\r' or '\n', the parser will do internal handlement, that is to save the field value into related variables according to different field tags.

According to the syntax of HTTP protocol, a blank line with only one character '\n' is used to separate the HTTP header and the HTTP body of an HTTP message. Therefore, if state `BEGINNING_OF_LINE` meets character '\n', it will do `END_OF_HEADER` handlement, including setting the final header fields' values and related routines for the next step after the whole MIME parsing. For example, for field 'Content-Type: text/html', the related routine should be 'HTMosaicHTMLPresent()' which is respon-

sible for displaying the document as an HTML file.

After the header parsing is finished, the state will be changed to `MIME_TRANSPARENT`. And the work at this state is just to read the following body characters into a certain buffer which is used for final display, this is the work done by handlement part `PUT_BODY_DATA_INTO_BUFFER`.

According to the new feature of HTTP/1.1, header field Content-type may include charset parameter. For example,

```
Content-type: text/html;charset=gb2312.
```

Therefore, MIME Parser has to be modified to handle the additional charset part.

In the `END_OF_HEADER` handlement part, modification has been made in function *HTMIME_put_character()* to separate the header field 'Content-Type' into two parts, one is the original part containing the format of the retrieved document, such as *"text/html"* or *"image/gif"*, the other part is the codeset part, the codeset information is saved into a new global variable `AfterMimeCodeset`, so that it can be used for later codeset conversion if necessary. The new global variable has the following data structure:

```
typedef struct my_codeset{
        int  font_size;
        char language[5];
        char codeset_name[20];
} MY_CODESET;

MY_CODESET AfterMimeCodeset;
```

Data structure *MY_CODESET* is defined in the new added header file my.h. In the above structure, item 'font_size' is corresponding to the menubar callback data, 'language' is used to identify different languages and 'codeset_name' stores the codeset information extracted from the HTTP response header. There are cases in which one language has several codesets, such as Chinese has codesets GB, Big5, CNS11643 and

etc. Codeset conversion is only carried out between different codesets within the same language. Codeset conversion between codesets belonging to different languages makes no sense.

After the header parsing is finished, the state is changed to be `MIME_TRANSPARENT`, and handlement part `PUT_BODY_DATA_INTO_BUFFER` takes the job. The processing in this part can be illustrated in the Figure 5.5.

There is another global variable `CurrentFont` which has the same structure of variable `AfterMimeCodeset`. It stores the current codeset setting by users or the local system. If MIME Parser extracts no codeset information from the HTTP response header, the body data of the HTTP response message following the HTTP header fields — the real data of the retrieved web document, are sent to `codeset detection` module (explained in Chapter 6) to accept codeset analysis. If `codeset detection` module cannot detect the codeset by investigating the binary code of the document, then the data are sent to the buffer directly without any codeset conversion. Otherwise, codeset conversion may be carried out depending upon the language information. On the other hand, if MIME Parser has gotten the codeset information, `AfterMimeCodeset` is not empty in this case, then something must be done to compare the language information holding by `AfterMimeCodeset` and `CurrentFont` first. If the two languages are the same, further comparison of two codeset_names are needed. If the two codesets are identical, no codeset conversion is done, otherwise codeset conversion from the codeset stored in `AfterMimeCodeset` to the one held in `CurrentFont` should be carried out at this stage. The converted document instead of the original one is sent to the buffer then.

No modification is made in handlement part `INTERNAL_HANDLE`.

Modification 3: HTML Parser

After MIME parsing of the HTTP response message, the body data of the retrieved web document are sent to HTML Parser for further analysis. The main task of HTML Parser is to separate the HTML document into different presentation units so that the

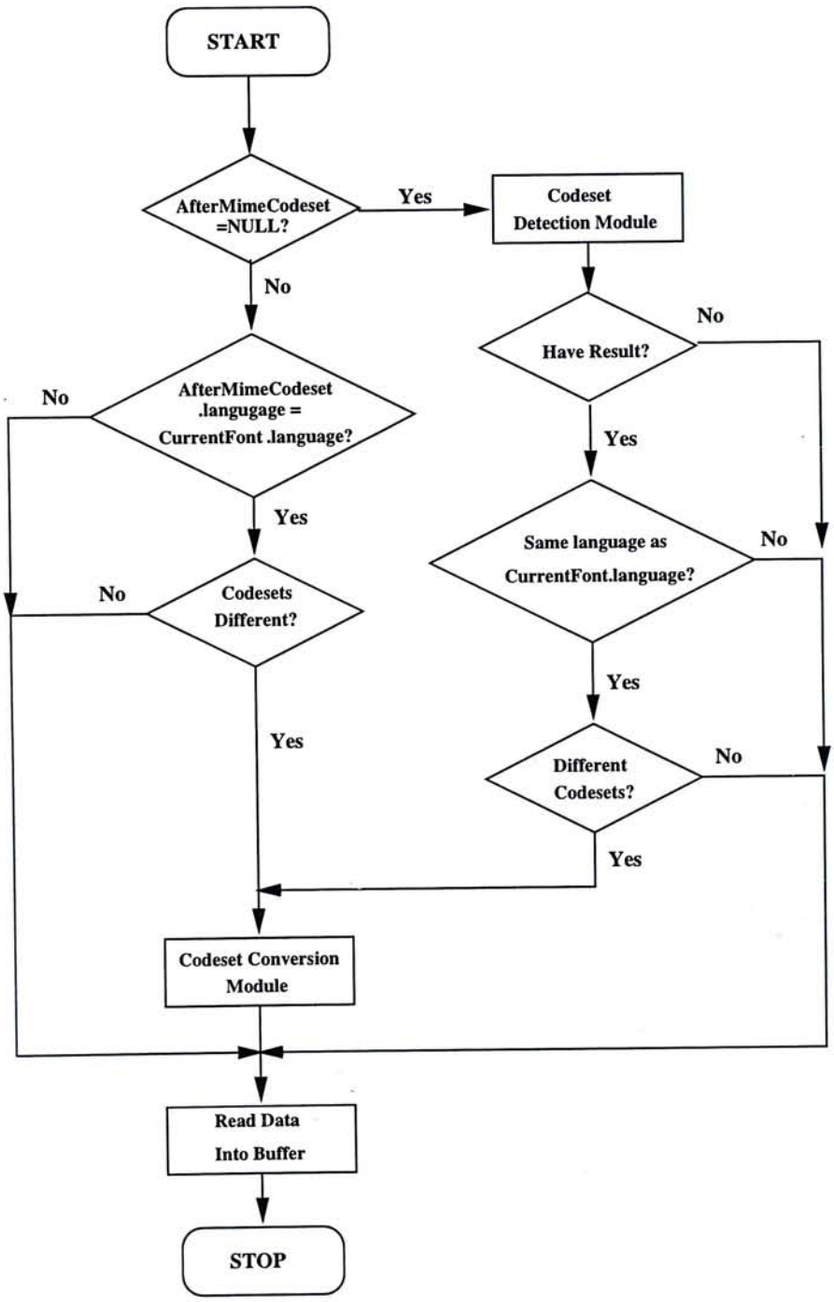

Figure 5.5: Codeset Conversion Handlement In MIME Parser.

Presentation Manager can display the document according to these units. Different kinds of units have different display strategies. If there is no codeset information in HTTP response header, and module 'Codeset Detection' cannot find the codeset of the body data of the web document, HTML Parser should investigate whether there are <LANG> tags inside the body data, if there are, HTML Parser extracts these information first, and then tries to convert all different codeset parts into the same target codeset if possible. After the codeset conversion, the converted data are separated by the normal HTML parsing modules, and the individual units are sent to the Presentation Manager for further display. The modification of HTML Parser is expected to be identical to what have done in the enhanced server.

Event Handler: User_Reset_Font

Another typical event handler `User_Reset_Font` which is just related to browser *interface* part has been modified also. After users reset the font option from the menubar, a menubar callback function *XmxCallback()* is invoked. The definition of this function is as follows:

```
XmxCallback(menubar_cb)
```

The main function of *XmxCallback()* is to let variable *mo_window *win* pointing to the current window, to get back the triggered event's corresponding numeric value and then to invoke corresponding event handling functions. For example, if users reset the font to simplified Chinese (the codeset is gb2312), the numeric value for this setting is *mo_charset_gb* which is in fact an integer number. Then the function `mo_force_font_change()` is invoked to do the right things. The definition of `mo_force_font_change()` is shown below:

```
mo_status mo_force_font_change(mo_window *win, int size)
```

The parameter *win* of type *mo_window* is pointing to the current window, and the size parameter is the numeric value of the triggered event. In this function, codeset

conversion has been added if the codeset of the current window is different from what users reset provided that they belong to the same language. The codeset conversion is done by a new function *conv_str()*. The definition of *conv_str()* is as follows:

```
void conv_str(char *sourcetext, char *desttext, char *fromcode,
              char *tocode, int textlen)
```

The *sourcetext* is a pointer pointing to the document which is currently displayed on the screen, the *desttext* is another pointer pointing to the buffer holding the converted stream data which will be displayed on the screen after the conversion. *fromcode* points to the original codeset of the document while *tocode* points to the target codeset to which the document will be converted. *textlen* tells the length of the source document before it is converted. The module *conv_str()* calls the Hanzix codeset conversion routines, and has the capability to handle several kinds of exceptions. Before carrying out codeset conversion, two new variables are needed to be defined in function *mo_force_font_change()*:

```
MY_CODESET pre, back;
```

pre is used to remember the current codeset setting before the codeset conversion while *back* gets the new setting after the event is triggered. If the two variables are not identical at the codeset part, and are identical at the language part, codeset conversion should be done.

5.4 Internationalization of Browser Interface

As a Chinese Internet access tool, the web browser's interface should be customized in a way that is convenient for Chinese users. For example, the interface for people in Mainland China should be simplified Chinese while the interface for people in Hong Kong or Taiwan should be traditional Chinese. The approach used to realize this objective is to adopt the concept of Internationalization (I18N for short) and Localization

(L10N for short) into the browser's *interface* part's development. Instead of providing different versions of the same browser interface for multiple codesets, only one version of browser interface is provided to users. Each time, users set up a particular local environment for their systems, the browser will then be customized to support only that particular environment at one time, the source code of the browser interface part doesn't need to be changed at all. This method facilitates the portability of the browser system.

In general, I18N refers to the approach of writing software in a way independent of the data and/or display information that are related to culture and language conventions. By applying I18N in software development, source code no longer needs to be modified or recompiled when porting to different regions where language or culture is different. If cultural dependent information, which is implicitly non-portable, can be isolated from the source code, only the external cultural-dependent information needs to be localized when the software is marketed to a different country or region. The process of customizing software for a particular language and/or culture is referred to as L10N. An internationalized software needs the localization process to provide language and culture related data although the localization process mostly involves preparing language and culture data in tabular forms outside the software.

To realize the internationalization of the browser interface, several *locales* which contain the culture and language related information should be set up, all program messages should be isolated from the program, and the program should be modified in an internationalized way which means the statements related to interface display should only describe the abstract information, the concrete data for a certain language or culture will be loaded dynamically when the program runs.

5.4.1 Locale

Locale is the facility provided by ISO POSIX. It specifies a particular language environment primarily containing a set of tables of predefined formats, defining various language-specific conventions, such as details of codesets used, date, time and mon-

etary representation formats. Access to data specified in a locale are through a set of POSIX interface functions. To use another language environment, a user needs to switch from one locale to another explicitly using a given designated function. Therefore the software can be coded completely independent of the locales. With POSIX support, several localized environments of different languages and customs can coexist in the same system, although only one locale can be active at any time in a given application.

The implementation of the enhanced browser's *interface* part is independent of any codesets. Before running the web browser, a user should specify the required locale and notify the browser by setting the `LANG` environment variable in the following format:

```
setenv LANG zh_TW
```

which means the *locale* is set to be traditional Chinese with codeset CNS 11643.

More than one locale has been built for Chinese language in our system. The relationship between language and locale is not necessarily a one-to-one mapping. The reason is that a single language such as Chinese might have different character sets (Simplified Chinese and Traditional Chinese). Besides, the same character set can have different codesets. For example, there are two popular codesets - Big5 and CNS for Traditional Chinese. Figure 5.6 illustrates the relationship between language and locale.

Since the default interface of our web browser is set to be English, there must be a locale for English. Three locales have been set up for Chinese. Table 5.1 shows the locale information.

5.4.2 Resource File

Our web browser has a separate resource file under each locale. The contents of this resource file include the locale-dependent information such as what fonts to be used, what default codeset to be used, what input method service to be connected and so

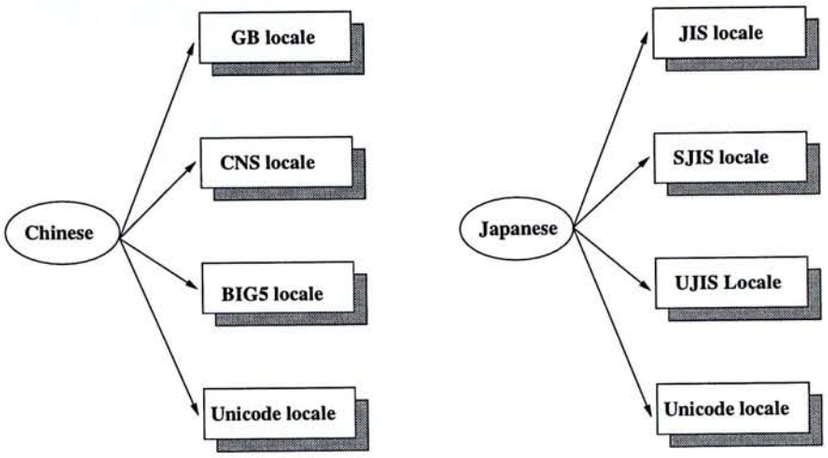

Figure 5.6: Relationship between language and locale

Table 5.1: Locales set up in our web system

Locale Name	Language	Region	Codeset
en_US	English	Western Countries	ASCII
zh	Simplified Chinese	Mainland China	gb2312
zh_TW	Traditional Chinese	Taiwan	CNS 11643
BIG5	Traditional Chinese	Taiwan, Hong Kong	Big5

on for the selected locale. If users want to customize their own locale environment, they only need to modify the resource file(s) instead of the source code of the browser. Figure 5.7 illustrates the relationship between locale and its resource file.

In our web browser system, all resource files are located under directory

```
/local/ciswb/.app-defaults/
```

For example, resource file for gb2312 codeset is:

```
/local/ciswb/.app-defaults/zh/Mosaic
```

and resource file for Big5 codeset is:

```
/local/ciswb/.app-defaults/zh_BIG5/Mosaic
```

All resource files for different *locales* have identical name - Mosaic, but the contents

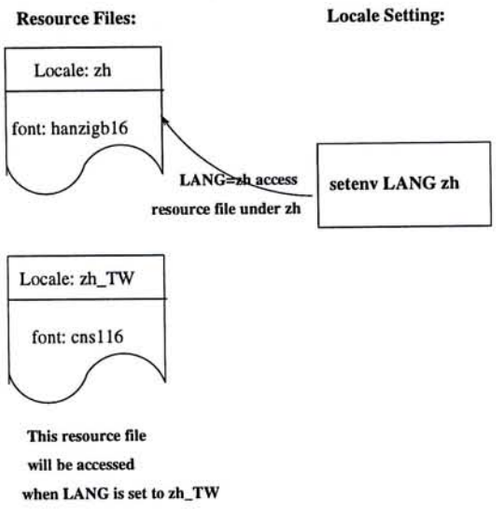

Figure 5.7: Relationship Between Locale and Resource File

are different from one to another. For example, in the resource file for GB *locale*, the font to be used is as follows:

```
Mosaic*XmPushButton*fontList: 8x16;hanzigb16st:
```

The same line in the resource file for BIG5 *locale* is as follows:

```
Mosaic*XmPushButton*fontList:
        -*-*-medium-r-normal-*-16-160-72-72-c-160-big5*-*
```

Each time, when users set a certain locale, the related information will be extracted from corresponding resource file and the interface of the browser will be changed to be different languages or different codesets according to the settings in the related resource file.

5.4.3 Message Catalog System

One of the most obvious need an internationalized system must fill is that of allowing a user to interact with the computer in his or her own language. However, uninternationalized code typically can produce messages in only one language because the messages are hard-coded into the program logic [31]. To make the browser *interface*

part to be internationalized, another facility provided by ISO POSIX called *message catalog* is adopted into our browser. Instead of hard-coding the text into the program, it is stored in a separate message catalog.

A message catalog contains program context data such as prompt information, help messages and so on. All these program messages are stored outside the program itself.

There are routines for creating, storing, accessing, and updating user-visible program text in a variety of languages. Users can translate the text into the languages of their choices. The result is multiple versions of the messages for a given program - one version per language and codeset combination [31]. The program then can interact with a user in any of these language/codesets. Adding new language/codeset is very easy, just supplying a new message catalog, the program code doesn't need to be changed.

The first step to build a message catalog system for our web browser is to isolate all program's messages into a message source file. The messages in the web browser includes all menubar titles, their options, buttons and so on. For example, menubar item *File* and its derivative options have following program messages:

```
File
  New Window          Clone Window        Open URL
  Open Local          Reload Current      Reload Images
  Refresh Current     Find in Current     View Source
  Edit Source         Save As             Print
  Mail To             CCI                 Close Window
  Exit Program
```

In the original version of the browser, all these messages are hard-coded in the program, so every time, only the English menubar can be shown to users. In our enhanced browser instead, all such kind of text data have been extracted from the program, and have been saved into message source files for different languages/codesets respectively.

The organization of a message source file is quite straightforward. Each line starts

with a message identification number, referred to as *msg id*. A *msg id* is either an integer or a mnemonic label which has been defined before. But in a message source file, either integer is adopted or a mnemonic label is used as a *msg id*, they can not be used simultaneously. If integers are used, they must be in ascending order, but they don't need to be consecutive. A single ASCII space or tab follows the *msg id*, and then the message text is listed. For example:

```
20 This is message number 20.\n
```

Message text within a source file can be in any language, and can contain any valid multibyte character. In our message catalog system, there are 3 versions for the same program messages for Chinese, one is for simplified Chinese gb2312 codeset, one is for traditional Chinese Big5 codeset, and another one is for traditional Chinese CNS codeset.

Some other options for building a message source file are listed below:

```
. Comments: Any line that begins with a dollar sign ($) followed by
            a space or tab is treated as a comment. For example:
            $ This is a comment

. Quoting mechanism: The message text can be surrounded with a quote
                     character of users' choice.

                     To designate a quote character, the source file
                     must include a line like:
                     $quote"
                     where the double-quote(") is the designated
                     character. For example:
                     25 "I am a student\n"

. Continuation character: Not all message text fits exactly on one line.
                          Use a backslash(\) to continue a message
                          across multiple lines. For example:
```

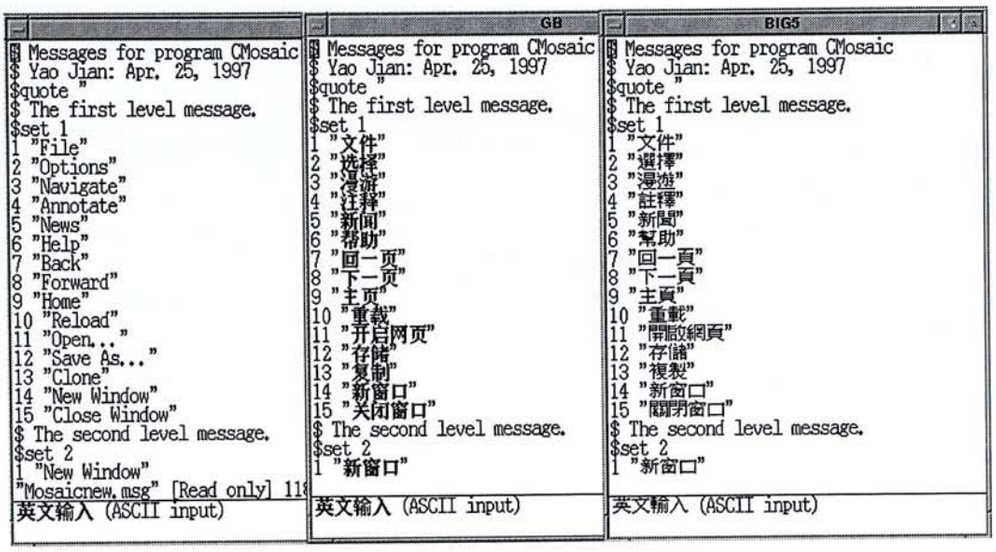

Figure 5.8: The Fragment of The Message Source File

```
12 This is a very very long \
message line.
```

```
. Message Sets: Message source files may be divided into sets. Sets are
                a useful tool to organize the messages in a more logical
                way. Related messages can be categorized into the same
                set. To designate a set, the following line must be
                included:
                $set n
                where n is the identification number of this set. If
                n is an integer, it must be 1(one) or higher.
```

Since there are nearly 100 messages in the browser's interface, *sets* are used to divide them into various groups. Set No. 1 includes the toppest level which includes the parent messages of menubars, including buttons, and set No. 2 contains the derivative messages from each menubar options, and the set number definition goes down in the similar way. The fragmental part of our message source file for English, Simplified Chinese (gb2312) and Traditional Chinese (Big5) versions are shown in Figure 5.8. The message source file for different languages/codesets adopts the same format, only the messages themselves are translated into related language/codeset instead of only English.

Table 5.2: File Types Related To Message Catalog System

File Type	Sample Name	Description
Source program	callmsg.c	Program source code with messaging calls
Message source file	callmsg.msg	source text of messages used in callmsg.c, source to *gencat*
Object catalog	callmsg.cat	created by *gencat*, an object file that callmsg accesses at its run time

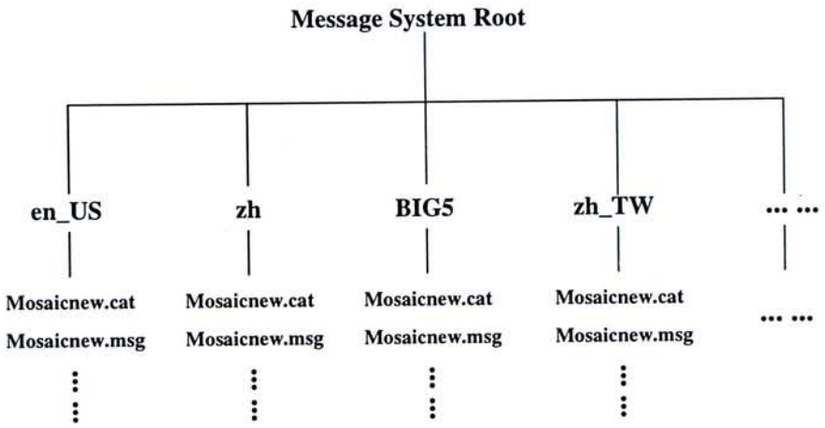

Figure 5.9: Message Catalog Directory Structure In Our System

Since message source files are only simple text files, they cannot be cited by program as they are not in binary format. Generating a message source file into its corresponding object catalog file is done by UNIX command *gencat*. The usage is:

```
gencat catfile msgfile
```

where *catfile* is the target object catalog and *msgfile* is the message source file. Table 5.2 illustrates the different types of files used in the message catalog system.

In our system, the message catalog files are stored in a separate directory as shown in Figure 5.9. The message file for the same program has a unique name: *Mosaicnew.msg* and *Mosaicnew.cat*. Different versions of message files are stored in a certain subdirectory under their corresponding *locale* name.

Once the message catalog files have been generated, the program can access the message data through the following 3 system calls:

- `catopen() for opening a version of a named message catalog as determined by the current locale.`
- `catgets() for retrieving a specific message string from that catalog, and`
- `catclose() for closing the named catalog.`

In our browser's *interface* part, function *mo_make_document_view_menubar()* is designed to make all menubars visable on the screen. And a new function *ingetmessage()* has been added into the former module to realize the access of the message data. The first step of function *ingetmessage()* is to open a message object catalog file:

```
catd = catopen("Mosaicnew.cat",0);
```

Then it tries to modify each menubar option's display by replacing the hard-coded text into the messaging calls. The original code for displaying menubar option *File* is:

```
static XmxMenubarStruct menuspec[] =
{
  { "File",       'F', NULL, NULL, file_menuspec },
 ... ...
```

The modified code is shown below:

```
menuspec[0].namestr = catgets(catd, 1, 1, "File");
```

This statement means the first element of the menubar can be obtained by calling function *catgets()* to get the first message of the first set (*msg id*= 1 and *set No.*=1) in message catalog file *Mosaicnew.cat.* String "File" is the default message in case no data is found in *Mosaicnew.cat.* All other interface messages are plugged into the program in the same way.

The environment variable named *NLSPATH* is used for defining the location of catalogs [31]. *NLSPATH* works much the same as other PATH-like variables. Table 5.3 shows some elements of an *NLSPATH* value. Before invokes our browser, the following two commands should be run first to indicate clearly the locale and the related message catalog file:

Table 5.3: Elements of an NLSPATH Value

Keyword	Meaning
%N	The value of the *name* parameter passed to *catopen()*
%L	The value of the full locale name
%l	The *language* part of the locale
%t	The *territory* part of the locale
%c	The *codeset* part of the locale
%%	A single % character

```
setenv LANG zh
setenv NLSPATH /local/MosaicBAK/message/%L/%N
```

The first line means the locale is set to be zh standing for simplified Chinese with gb2312 codeset. The second line tells the program to get the message catalog file with the full path: */local/MosaicBAK/message/zh/Mosaicnew.cat.*

5.5 Experiment Result

Figure 5.10 shows the two versions of Chinese interface of our browser: one is for simplifiled Chinese with gb2312 codeset, the other is for traditional Chinese with Big5 codeset. Each time, before the browser is invoked, users have to set up the *locale* first so that the browser can access the right message catalog when it runs and plug in the related data accordingly.

For the gb2312 interface, users have to set up *locale* in the following format:

```
setenv LANG zh
```

And to set up for Big5, the line should be:

```
setenv LANG BIG5
```

Figure 5.11 shows the on-line codeset conversion done by our enhanced browser. The document named *exhi.htmlgb* is written in simplified Chinese. When users reset the

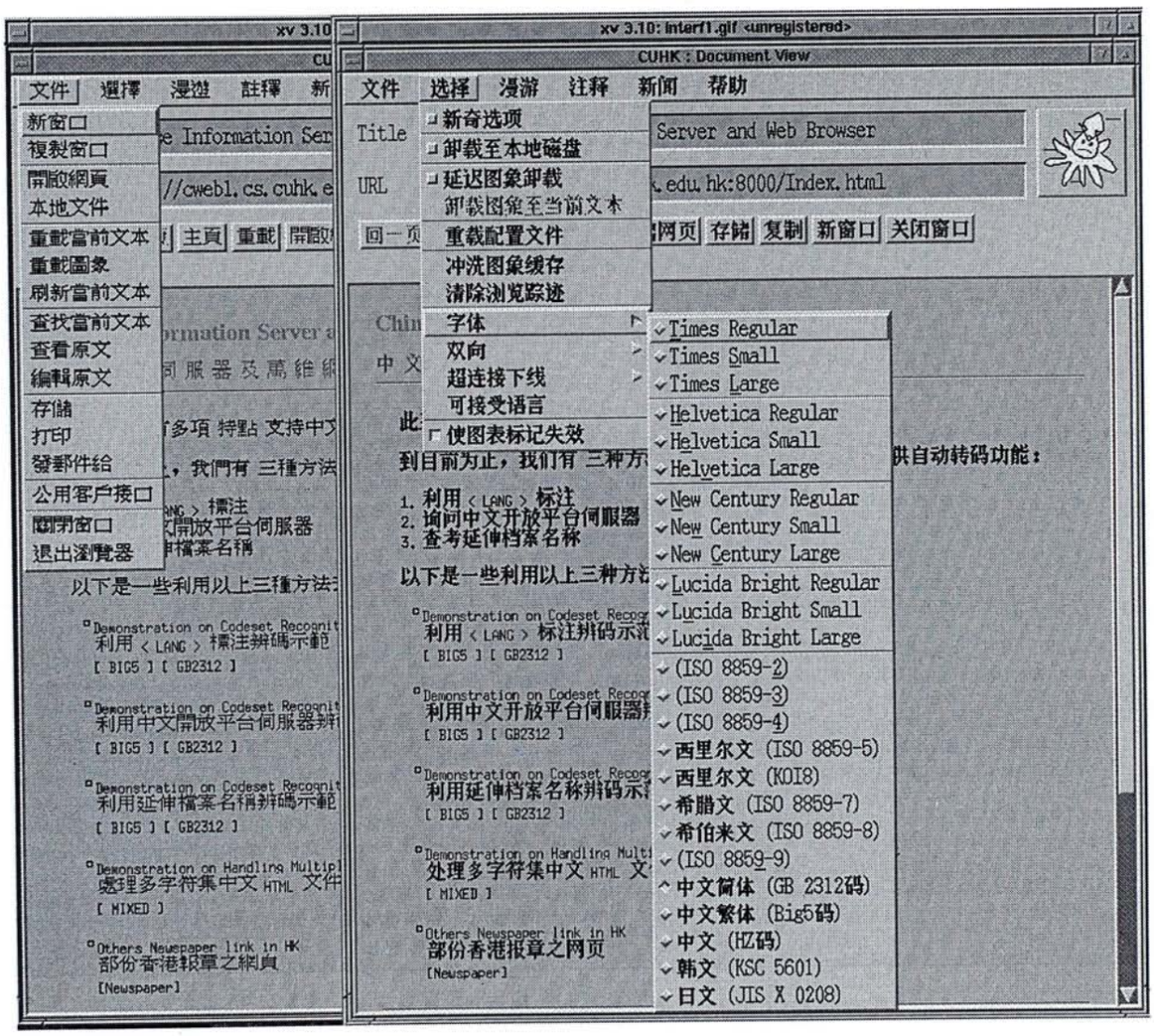

Figure 5.10: The Chinese Interface of Our Browser

Figure 5.11: On-line Codeset Conversion Done By The Enhanced Browser

font to be Big5, the browser does the on-line codeset conversion, so that the converted version in traditional Chinese of the same document is displayed on the screen.

As to the facility of data type negotiation in our enhanced browser, Figure 4.4, Figure 4.5 and Figure 4.6 in Chapter 4 are all examples to demonstrate it. Without the codeset announcement from the enhanced browser, the data type negotiation cannot be completed only by the enhanced server.

Chapter 6

Another Scheme - CGI

Through the above discussion, it is known that a system with both enhanced browser and server can realize Chinese text data access with on-line codeset conversion and can avoid duplication of Chinese documents. However, since the enhanced browser is developed based on Mosaic, it cannot overcome the drawbacks inherited from Mosaic, such as Mosaic doesn't support some latest features like: Java, Java script, CGI and etc.. As we know, among various existing browsers, Netscape is very popular and has occupied a large proportion of market on Unix platform. It is not wise and applicable to force Internet users to use our enhanced browser instead of the one which is preferred by themselves. In such case, something must be done to reach the same goal without the help of our enhanced browser. A new scheme is devised to fulfill special requirements for Chinese information processing between a normal web browser(in our project, we choose Netscape as such a normal web browser) and a normal web server (in our current development, our own web server is taken as such a normal web server, in the next stage, any web server can be accessed using our approach). This scheme adopts one of the latest web technique — Common Gateway Interface which can pass the user side information to the server, and vice versa. With cgi technique, data type negotiation can be carried out without the need to enhance the browser/server to support the new features of HTTP/1.1, instead, all works including codeset conversion are done by CGI itself. In our system, CGI runs as a back-end program on our web server. Users use Netscape to access files on our web server.

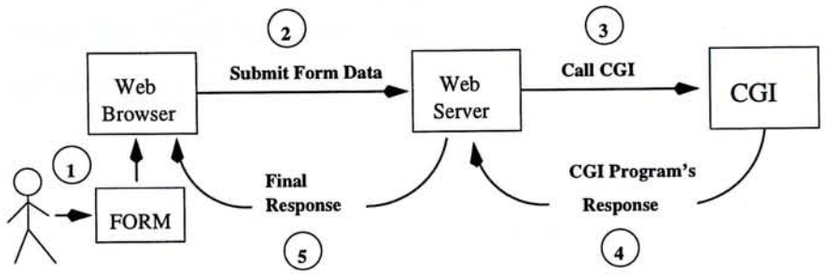

Figure 6.1: The Operation Procedure of CGI.

6.1 Form and CGI

Data type negotiation needs both web browser and server to involve in the communication of data types they support during each round of information retrieval. Since normal web browser and web server don't have ways to tell such data type information, an interactive interface must be established to let users to input such information manually, and these data should be sent to the server for further processing to fulfill users' requirements. The interface part can be realized using HTML's form, and the passing of the inputted data and further handlement at the server side can be carried out by the cgi program. In fact, the name of the cgi program should be indicated in the form, and they cooperate to carry out related tasks. Figure 6.1. illustrates the operation procedure of CGI.

The Common Gateway Interface(CGI), forms are generally used for two purposes: data collection and interactive communication. Users can conduct surveys or polls, and present registration or online ordering information through the use of forms. They are also used to create an interactive medium between the user and the web server. For example, a form can ask the user to select a document out of a menu, whereby the server returns the chosen document [15]. The second usage of forms and cgi is adopted in our scheme. An interactive interface has been designed for users to select different codesets through clicking a certain button. And the cgi program tries to get the document and convert it into the codeset preferred by users, and at last send this converted document back to the client.

A form consists of two distinct parts: the HTML code and the CGI program.

HTML tags create the visual representation of the form, while the CGI program decodes (or processes) the information contained within the form [15]. Here is an example of a simple form:

```
<FORM ACTION="http://cweb1:8001/demo.cgi" METHOD="GET">
```

The <FORM> tag starts the form while </FORM> ends it. The two attributes within the <FORM> tag (ACTION and METHOD) are very important. The ACTION attribute specifies the URL of the CGI program that will process the form information, here cgi program named *demo.cgi* at site *http://cweb1:8001* will be invoked after users input data into this form and click submit button. The METHOD attribute specifies how the server will send the form information to the program. POST sends the data through standard input, while GET passes the information through environment variables. If no method is specified, the server defaults to GET [15].

Most form elements are implemented using the <INPUT> tag. The TYPE attribute to <INPUT> determines what type of input is being requested. Several different types of elements are available: text and password fields, radio buttons, and checkboxes. In our implementation, radio buttons are devised to ask for users' selection. Here are the HTML codes:

```
<form action="http://cweb1:8001/cgi/demo.cgi">
GB:      <input type="radio" name="codeset"  value="GB">
Big5:    <input type="radio" name="codeset"  value="Big5">
CNS:     <input type="radio" name="codeset"  value="CNS">
Unicode:<input type="radio" name="codeset"  value="UNICODE">
         <input type="submit" value="Submit">
         <input type="reset"  value="Reset">
</form>
```

The corresponding result displayed by Netscape is shown in Figure 6.2. The above code presents four choices for users: GB (simplified Chinese with gb2312 codeset),

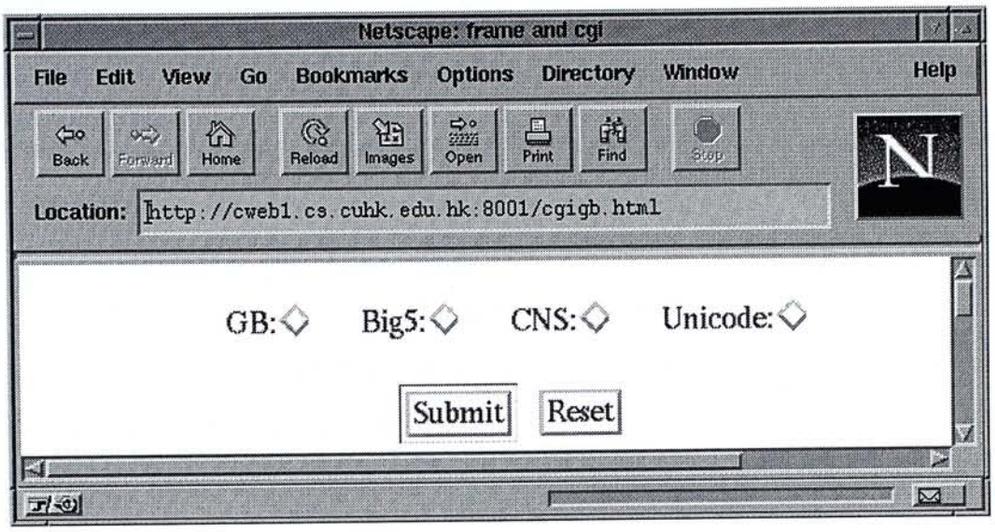

Figure 6.2: The Interactive Interface Designed For Codeset Announcement

Big5 (traditional Chinese with Big5 codeset), CNS (traditional Chinese with CNS or x-euc-tw codeset defined by Netscape) and Unicode (Unicode codeset which covers multiple languages).

Two more important "types" of the <INPUT> tag are *Submit* and *Reset*. Nearly all forms offer these two buttons. The *Submit* button sends all of the form information to the CGI program specified by the ACTION attribute. Without this button, the form will be useless since it will never reach the CGI program. The *Reset* button clears all the information entered by the user. Users can press *Reset* if they want to erase all their entries and start all over again [15].

After users input data and click *Submit* button, in the above example, users may click button GB and *Submit*, then the codeset of GB is transferred to the server and the server tries to execute the CGI program.

CGI is the part of the web server that can communicate with other programs running on the server. With CGI, the web server can call up a program, while passing user-specific data to the program (such as what host the user is connecting from, or input the user has supplied using HTML form syntax). The program then processes that data and the server passes the program's response back to the web browser. When a CGI program is called, the information that is made available to it can be broken

into two groups:

. Information about the client, server, user and additional pathname.

. Form data that the user supplied.

Most information about the client, server or user is placed in CGI environment variables. And extra path information is placed in environment variables. Form data is either incorporated into an environment variable, or is included in the "body" of the request depends upon which method is used in the form [15]. If the GET method is used, the query string is simply appended to the URL of the program when the client issues the request to the server. This query string can then be accessed by using the environment variable QUERY_STRING [15]. If the POST method is used, the form data will be included in the HTTP request message's body, and the main advantage to the POST method is that query length can be unlimited. Since the form data of our system is just the codeset information, the query length is short enough to be safely handled by the client and server, therefore, GET method is chosen in our form/CGI operation.

Much of the most crucial information needed by CGI applications is made available via UNIX environment variables. Table 6.1 shows some useful environment variables which can be used by CGI programs. Among all the above environment variables, REQUEST_METHOD and QUERY_STRING are two important environment variables which are used by our CGI program. REQUEST_METHOD tells CGI programs what is the method the form used (GET or POST), and QUERY_STRING provides the query data entered by users. For example, the content of QUERY_STRING may be "codeset=GB" which means users choose GB as their preferred codeset. After CGI program gets the form data, it has to decode the data and interpret them correctly. The following is the algorithm which is used by CGI program when it decodes the input form data [15]:

1. Determine request protocol(either GET or POST) by checking the REQUEST_METHOD environment variable.

Table 6.1: Some Useful Environment Variables

Environment Variable	*Description*
SERVER_NAME	The hostname or IP address of the server.
SERVER_SOFTWARE	The name and version of the server software that is answering the client request.
SERVER_PROTOCOL	The name and revision of the information protocol the request came in with.
SERVER_PORT	The port number of the host on which the server is running.
REQUEST_METHOD	The method with which the information request was issued.
PATH_INFO	Extra path information passed to a CGI program.
QUERY_STRING	The query information passed to the program. It is appended to the URL with a "?"
REMOTE_HOST	The remote hostname of the user making the request.
CONTENT_TYPE	The MIME type of the query data, such as "text/html".
CONTENT_LENGTH	The length of the data (in bytes or the number of characters) passed to the CGI program through standard input.
HTTP_ACCEPT	A list of the MIME types that the client can accept.
HTTP_USER_AGENT	The browser the client is using to issue the request.
HTTP_REFERER	The URL of the document that the client points to before accessing the CGI program.

Table 6.2: Valid HTTP Headers for CGI programs

Header	*Description*
Content_length	The length(in bytes) of the output stream. Implies binary data.
Content_type	The MIME content type of the output stream.
Expires	Date and time when the document is no longer valid and should be reloaded by the browser.
Location	Server redirection (cannot be sent as part of a complete header).
Pragma	Turns document caching on and off.
Status	Status of the request (cannot be sent as part of a complete header).

2. If the method is GET, read the query string from QUERY_STRING and/or the extra path information from PATH_INFO.

3. If the method is POST, determine the size of the request using CONTENT_LENGTH and read that amount of data from the standard input.

4. Split the query string on the "&" character, which separates key-value pairs (the format is key=value&key=value...).

After decoding the input form data, CGI program will carry out related processing to complete the requirements from users. Instead of returning a static document, the server executes the CGI program and returns its output. Since any HTTP response message must include header information, these information should be composed by CGI program either partially or completely. Table 6.2 shows some useful HTTP headers for any CGI program [15]. HTTP response header Content-type has charset parameter which is defined in HTTP/1.1. For example, Content-type: text/html; charset=big5. In our system, if automatic codeset conversion has been carried out by the CGI program, it should include the converted codeset information into this response header so that the browser like Netscape 3.0 can interpret the body data correctly.

If the CGI program composes all HTTP response headers, the server will pass

its output directly to the client browser, otherwise, the server will get the output of CGI program and add some other HTTP response headers like the server information, status code, etc. to the original CGI output. In our system , the CGI program only composes the HTTP response header in a partial manner.

6.2 CGI Control Flow

To realize data type negotiation between web browser and web server, an interactive interface using form has been implemented. This interface provides four selections for users to choose: GB, Big5, CNS and Unicode. GB stands for simplified Chinese while Big5 and CNS are two different codesets for traditional Chinese, and Unicode is the coming codeset which can cover nearly all languages in the world. After users select a codeset from the interface, the server will invoke the CGI program to do the related works, including decoding the input form data, fetching the retried document located at the server site, automatic codeset detection of the retrieved document, automatic codeset conversion of the document, composing HTTP response message containing the codeset information. Figure 6.3. shows the control flow of our CGI program.

6.3 Automatic Codeset Detection

In the figure of CGI control flow, it is known that automatic codeset detection is the basis for the automatic codeset conversion. Since there are quite a few different codesets for Chinese characters, and the code range of most of the codesets are overlapped, it is not easy to distinguish one codeset from another only by investigating the binary code. In our system, we only confine ourselves on the automatic codeset detection between GB and Big5, i.e., we always assume that the retrieved document is encoded in either GB or Big5.

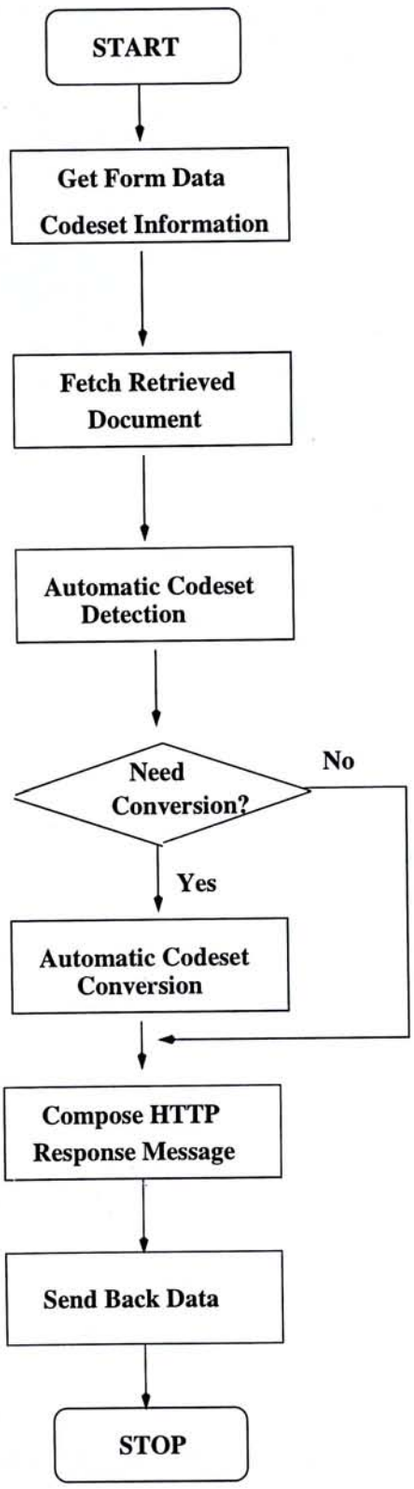

Figure 6.3: Control Flow of CGI Program.

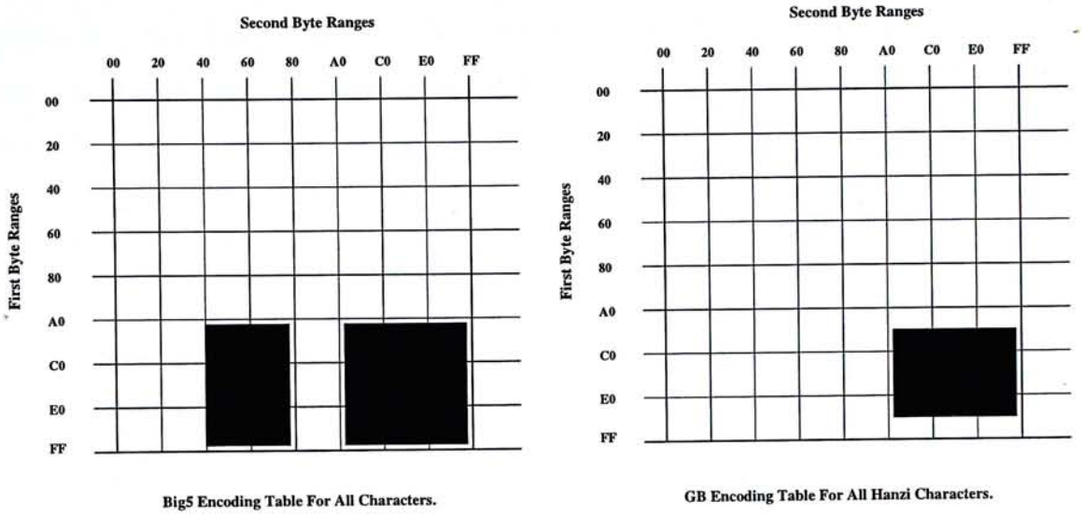

Figure 6.4: Big5 and GB encoding table.

6.3.1 Analysis of code range for GB and Big5

As an ideographic language, Chinese character system has thousands of characters. Because of this, an eight-bit code set with its maximum of 256 characters obviously is inadequate. Chinese language needs to use 16 bits or more for most characters. The standard codeset of simplified Chinese is GB 2312 which covers 6,763 hanzi characters, and the popular-used codeset for traditional Chinese is Big5 which covers 13,053 hanzi characters. Both of them adopt two bytes representation scheme, i.e., two bytes per character. And the MSB(Most Significant Bit) of the first byte is always 1 for both GB and Big5. For GB, the MSB of the second byte is also set to 1 while the MSB of the second byte of Big5 may be 1 or 0. Figure 6.4 shows the code range for both GB and Big5.

From the above encoding table, we can get the following observations:

1. Whether a character is GB or Big5 codeset, the MSB of its first byte must be 1, this is a tool to distinguish an ASCII character from a Chinese character.

2. The range for the first byte of Big5 is from A1 to FE while the range for the first byte of all hanzi characters of GB is from

```
   B0 to F7; The range for the second byte of Big5 has two sections:
   one is from 40 to 7E, the other is from A1 to FE while the range
   for the second byte of GB is from A1 to FE.

3. The code value for all hanzi characters of GB is a subset of the
   code value for all characters of Big5.

4. If a Chinese character's code meets the following conditions:
   first byte in [A1, AF] or first byte in [F8, FE], then it must be
   a Big5 character.

5. If a Chinese character's code meets the following condition:
   second byte in [40, 7E], then it must be a Big5 character.
```

6.3.2 Control Flow of Automatic Codeset Detection

The control flow of automatic codeset detection between GB and Big5 is illustrated in Figure 6.5. The detection module first checks the MSB of an input character, if it is 0, then it means that the character is an ASCII one, otherwise further analysis is needed in the following steps. If the MSB of all characters in the file is 0, then it must be an ASCII file. If any BIG5 special character is detected out, then the program regards this file as a Big5 one, this is based on the assumption that any readable and meaningful Chinese document encoded in Big5 must have characters whose code values are in different sections. If a character's code value is in the overlapped section, then just ignore it, and investigate the next character. If all the characters have been checked, and the MSB of characters is 1, but all characters are not Big5 special ones, then the program regards the file as a GB encoded one.

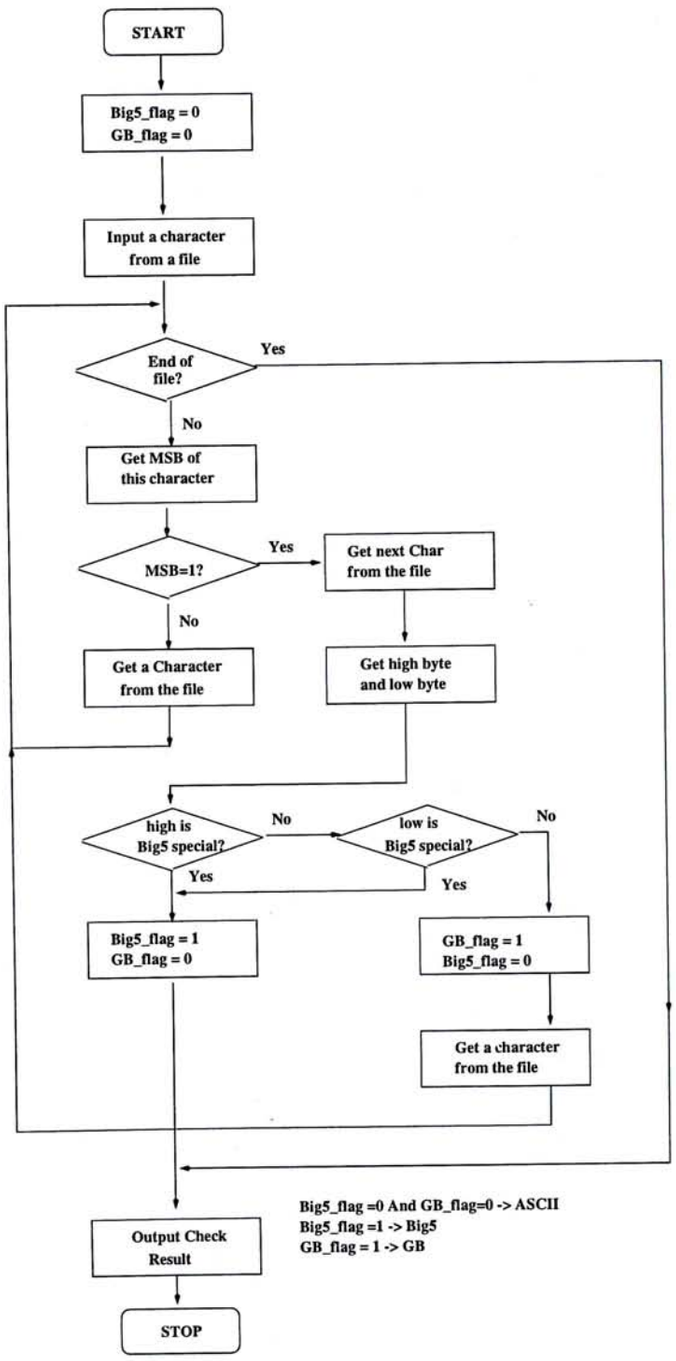

Figure 6.5: Control Flow of Automatic Codeset Detection.

6.4 Experiment Results

Figure 6.6. shows the experiment result of our CGI tests. The interactive interface site is at http://cweb1:8001/cgib5.html. This file is originally written in Big5, after users click GB, the returned document has been converted into GB format. This avoids the duplication of documents and provide friendly Chinese information access environment.

Another example is shown in Figure 6.7. The original codeset of file cgigb.html is gb2312, after users click Big5 button, the converted document encoded in Big5 is displayed on the screen.

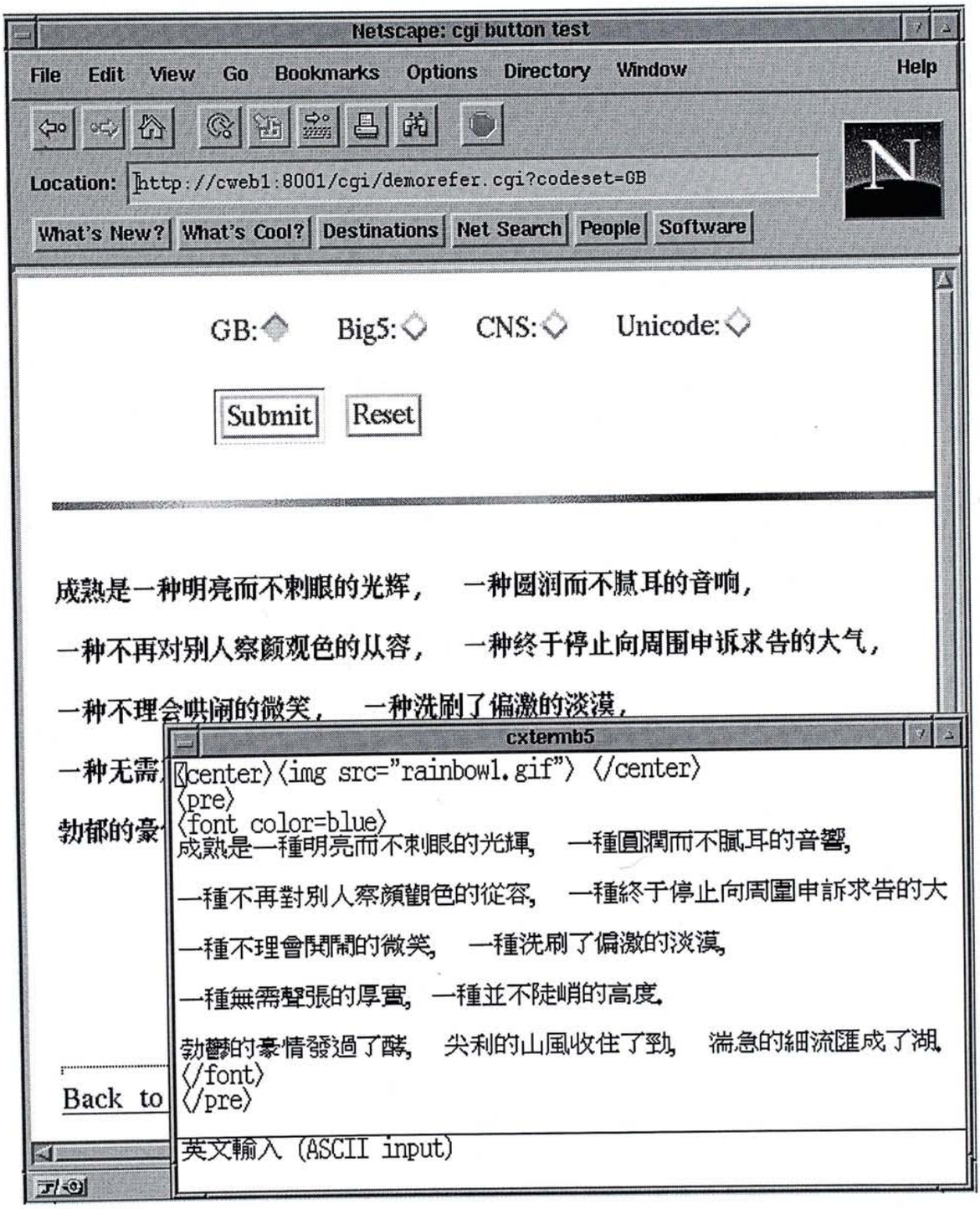

Figure 6.6: Codeset Conversion From Big5 To GB

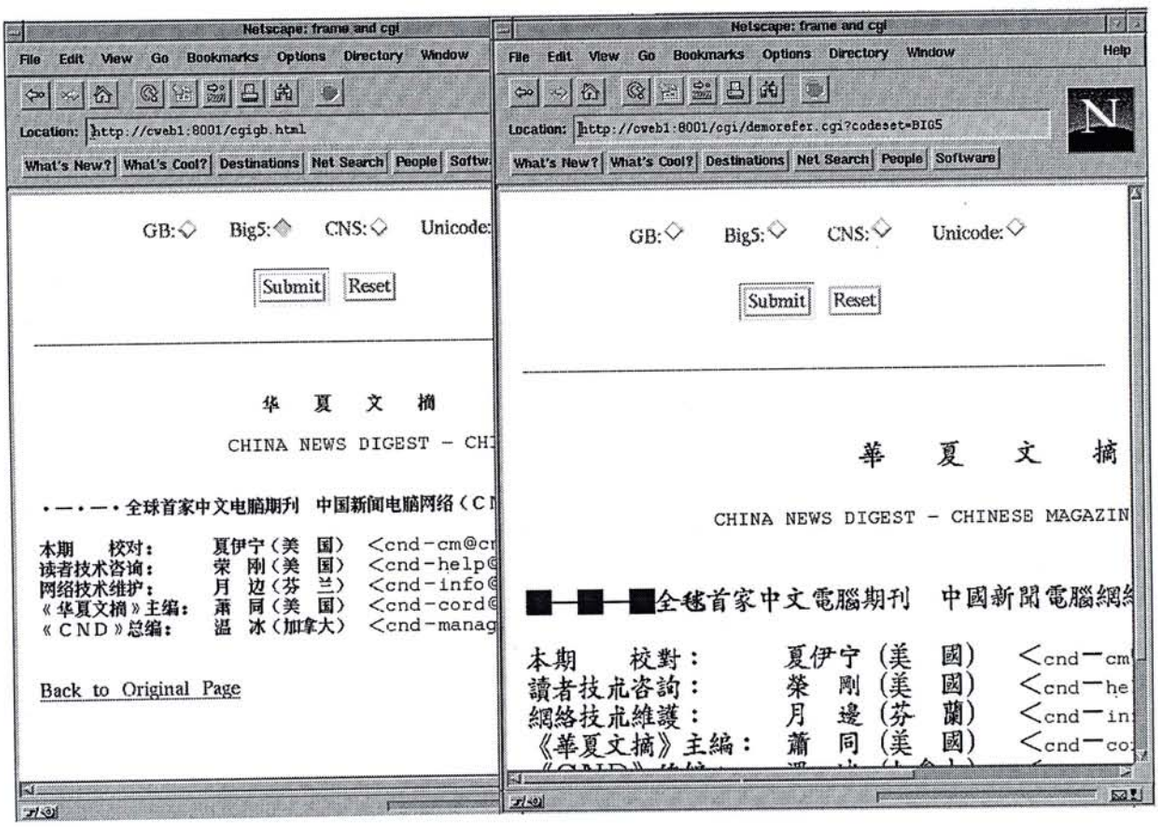

Figure 6.7: Codeset Conversion From GB To Big5

Chapter 7

Conclusions and Future Work

Exchange and manipulation of Chinese text information through World Wide Web are in great demand in Mainland China, Hong Kong, Taiwan and other places where Chinese characters are used. To provide a friendly environment for users to fetch, exchange and process Chinese text data via Internet, we design a new World Wide Web system to fulfill the special requirements for Chinese information access through Internet. Comparing with most of the current web servers/browsers which support Chinese text data processing, the main idea of our system is to introduce the codeset announcement mechanism which is realized through the data type negotiation between the web server and the web browser, and to carry out on-line automatic codeset conversion transparently to web users. With the on-line codeset conversion, duplication of Chinese documents are avoided. Also the internationalized user interface of the web browser provides a very convenient and friendly environment for users to do Internet access in their local familiar way, either using traditional Chinese or simplified Chinese without the need to match the codeset of the documents being retrieved from the server.

Our web system consists of three main components: the Chinese web server, the internationalized browser and an enhanced proxy server. All of them are developed under X-Open environment. The component integration approach is used in our architectural design, where each component is independent and reusable, and all components can

act in flexible combination to provide services under different situations. The development of our web system makes the Chinese text data processing over Internet more convenient and popular. The component integration concept is applicable to both Unix platform and PC platform. This approach can also apply to documents access between different languages. The codeset converter can be replaced by an intelligent language translator. In this way, document written in Japanese may become accessible to users who doesn't know Japanese.

7.1 Current Status

Currently, our enhanced server is capable of doing automatic codeset conversion and can handle data type negotiation with the enhanced browser. There are 3 methods for the enhanced server to detect the codeset of a retrieved document: with the <LANG> tag defined in HTML/3.0, with the help of I-Hanzix server and with file extension. Our enhanced server can handle a multiple codesets document and tries to convert the various codeset text data into the same target codeset.

Our enhanced browser has built the codeset announcement mechanism into the system, and it can properly parse the response message sent back by the server and display the document with a proper font without users' intervention. Also, if users want to reset the font while reading a document, the enhanced browser can carry out the on-line codeset conversion immediately. The browser's interface has been modified in an internationalized way, so that users can access server information either in traditional Chinese or simplified Chinese.

In order to make our system more flexible, a new scheme with CGI technique is devised to fulfill the same requirements completed by our enhanced web server and web browser system. It works fine for normal web browser - Netscape and a normal web server - our own server. It provides an interactive interface for users to manually select various codesets. The CGI does the automatic codeset conversion transparently to client users.

The core part of the enhancement of our server and browser are finished. Currently we are working at the proxy server located at the same side of the browser. The proxy server with the automatic codeset conversion functionality will be the bridge between the enhanced browser and any normal server. The cache of the proxy server saves both the original and converted versions of the web document to speed up the retrieval of Chinese documents.

7.2 System Efficiency

The implementation of our web system is based on the following criteria:

1. Modularity: all new functions are coded into individual modules respectively.

2. Minimum Coupling: all modules added have high independence, such as module conv_str() makes the input/output data as parameters which is least dependent of outside influences.

3. High Cohesion of Module: each module performs a single well-defined task: for example, function mo_get_accept_charset() gets the codeset information accepted by the browser.

4. Good Readability: We use both internal and external documentation to ensure the good readability of our program. The methods used in internal documentation include using of good name, such as function conv_str() and variable AfterMimeCodeset[], using comments such as function and module header comments and statement explanation comment. The methods used in external documentation include writing phased implementation report and programmer's guide, etc..

5. Programming Reliability: Our system has capability to tolerate faults in some extent when conducting MIME parsing, and it provides excepting handling in codeset conversion routines.

The platform our web system running on is Sun Sparc Machine with the Operating System System V Release 4.0. The time to access Chinese documents needed to be converted on our server is less than one second, and the time for on-line codeset conversion by the browser is less than one second which are both acceptable by web users. Since our web server keep only one codeset version of Chinese document, comparing with other servers which duplicate Chinese documents, the proportion on space saving increases directly as the number of supported codesets increases.

We adopt DLL (Dynamic Link Library) technique when linking the codeset conversion routines which saves resources and the library can be shared by all programs. Since the library is not linked at the compile time, the object code is smaller, the run speed is fast than usual.

7.3 Future Work

As personal computers become more and more popular in ordinary families, Internet access will be conducted by PCs at most of the time instead of workstations in the future. Therefore, we must ported our web system from Unix platform to PC platform in the next step. Since most of the commercial web browser's source code is not open to public, and buying them is very expensive, ways have to be found to develop a new web system based on the popular web browser to reach the objective proposed by us, such as using Plug-ins supported by Netscape, CGI technique, Java Script and so on.

As a web system for Chinese information processing, a search engine will be provided in the future for users to search Chinese information encoded in different codesets. For example, users want to access Chinese word "中国", the search engine must be able to find out all documents containing this Chinese word encoded in all possible codesets: gb2312, Big5, CNS, Unicode, etc. Since users' main concern is related to the semantic meaning of the Chinese word rather than the binary code representation in computer. Therefore no matter what codeset the Chinese word is encoded in, they should be regarded as the search result.

Currently our CGI system can only work with our own server, that is users can only access the files located on our own server. This restriction may not be accepted by users if the information they want to access is located elsewhere. Therefore the next objective is to improve our system so that users can access any remote web server through the interactive interface and our CGI program. This needs the CGI to connect with the remote server by itself through establishing an HTTP connection with the remote server, and the CGI is also responsible for collecting the data sent back from the server and conducts codeset detection and conversion if needed.

The accuracy of automatic codeset detection is very important. If the detection gives a wrong codeset result, the codeset conversion routine will generate garbage data. Currently, we can only detect two Chinese codesets: gb2312 and Big5. This is not sufficient. As the difficulty of detecting Chinese codeset from one to another, semantic approach is under our consideration. When a binary code is in the overlapped range of several codesets, the context data will be taken into consideration, and if the word before or after the current word can make meaningful phrases or structures, then the right codeset will be detected out.

All codeset conversion in our system are done through invoking related codeset conversion routines developed in Hanzix system. When 1-to-N mapping is encountered, the routine will either provide all possible output to users, or just select the first one by default. In the next step, some extent of intelligence will be added into the codeset conversion routine so that the right conversion will be given out by the routine directly without users' intervention.

Bibliography

[1] http://www.sai.msu.su/untpdc/training/net_connect/www.html, October 1995.

[2] N. Borenstein and N. Freed. Mime (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies (rfc 1521), September 1993.

[3] Lu Chin, Lee Kin Hong, Nui Pui Tak, Yao Jian, and Wong Man Fai. Technical report: A chinese internet information server with automatic codeset conversion functionalities, March 1996.

[4] Douglas E. Comer. *Internetworking with TCP/IP Volume 1: Principles and Protocols and Architecture 2nd edition.* Printice-Hall, 1991.

[5] General Electric Company. *Software Engineering Handbook.* New York: McGraw-Hill, 1986.

[6] China News Digest. http://www.cnd.org:8028/hxwz/.

[7] Dale Dougherty, Richard Koman, and Paula Ferguson. *The Mosaic Handbook for the X Window System.* O'Reilly & Associates, 1994.

[8] Charles. Easteal. *Software Engineering Analysis and Design.* McGraw-Hill, 1989.

[9] R. Fielding, H. Frystyk, and T. Berners Lee. Hypertext transfer protocol http/1.1 (http working group, internet draft), January 1996.

[10] William B. Frakes. *Software Engineering In the UNIX/C Environment.* Prentice Hall, 1991.

[11] Ian S. Graham. *The HTML source book.* John Wiley, 1995.

[12] Hanzix Work Group. The hanzix open system. In *Proceedings of International Conference on Chinese Computing '94*, Singapore, June 1994.

[13] The Hanzix Work Group. Hanzix open systems - codeset announcements, its principles and implementations, July 1995.

[14] The Hanzix Work Group. Technical report: The interim hanzix open systems - codeset conversion functionalities, July 1995.

[15] Shishir Gundavaram. *CGI Programming on the World Wide Web.* O'Reilly & Associates Inc., 1996.

[16] Lam Yuk Helen. Technical report - a chinese input method system for x-window applications - the open systems approach, September 1996.

[17] Borka Jerman-Blazie. Tool supporting the internationalization of the generic network. *Computer Networks and ISDN Systems*, pages 429–435, 1994.

[18] Yao Jian. Technical report - mosaic analysis and browser architecture design, July 1996.

[19] Yao Jian and Paul Pang. Fsm mime parser analysis and modification, October 1996.

[20] Brian Kelly. Becoming an information provider on the world wide web. *Computer Networks and ISDN Systems*, pages 353–360, 1994.

[21] T. Berners Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol http/1.0 (http working group, internet draft), September 1995.

[22] X/Open Company Limited. *Internationalization Guide.* X/Open Company Limited, 1990.

[23] Cricket Liu, Jerry Peek, Russ Jones, Bryan Buus, and Adrian Nye. *Managing Internet Information Services.* O'Reilly & Associates, Inc., December 1994.

[24] Chin Lu and Kin Hong Lee. Project draft — a chinese internet information server and the server access software, 1995.

[25] Qin Lu, Kin-Hong Lee, Jian Yao, and Man-Fai Wong. The design of a chinese world wide web server and an internationalized browser. In *Proceedings of the Seventeenth International Conference on Computer Processing of Oriental Languages*, Hong Kong, April 1997.

[26] Ken Lunde. *Understanding Japanese Information Processing.* O'Reilly & Associates, Inc., 1993.

[27] Ken Lunde. ftp://ftp.ora.com/pub/examples/nutshell/ujip/doc/cjk.inf cjk.inf version 2.1, July 1996.

[28] Ari Luotonen and Kevin Altis. World-wide web proxies, April 1994.

[29] Henrik Frystyk Nielsen and Hakon W. Lie. Towards a uniform library of common code - a presentation of the cern world-wide web library. *Computer Networks and ISDN Systems*, pages 13–23, 1995.

[30] Lawrence Nui Pui Tak. Technical report: Modifications of httpd 3.0, July 1996.

[31] Sandra Martin O'Donnell. *Programming for the world - A guide to internationalization.* Prentice-Hall, 1994.

[32] Ross Paterson. http://ifcss.org:8001/www/pub/software/info/cjk-codes/ cjk-codes.txt, February 1995.

[33] Bryan Pfaffenberger. *World Wide Web Bible.* MIS Press, 1995.

[34] Dave Raggett. Hypertext markup language specification version 3.0 (http working group, internet draft), March 1995.

[35] Paul E. Renaud. *Introduction to Client/Server System - A practical guide for systems professionals.* New York: Wiley, 1993.

[36] Ricky and H.T.Chan. A new unicode-based www gateway for browsing multilingual information on the internet. In *Eight International Unicode/ISO 10646 Conference*, Hong Kong, April 1996.

[37] Ian Sommerville. *Software Engineering.* Addison Wesley, 1996.

[38] Toshihiro Takada. Multilingual information exchange through the world-wide web. *Computer Networks and ISDN Systems*, pages 235–241, 1994.

[39] Andrew S. Tanenbaum. *Computer Networks.* Printice-Hall, 1989.

[40] TAKADA Toshihiro. http:// www.ntt.co.jp / mosaic-l10n/ readme.html.

[41] Matthijs van Doorn and Anton Eliens. Integrating applications and the world-wide web. *Computer Networks and ISDN Systems*, pages 1105–1110, 1995.

[42] F. Yergeau, G. Nicol, G. Adams, and M. Duerst. Internationalization of the hypertext markup language (network working group, internet draft), February 1996.

Appendix A

Programmer's Guide

This chapter describes the modification of the source code for enhancing the web browser. The internationalized browser is developed based on the Mosaic 2.6 which is a free software located in the public domain. There are around 70,000 lines of code in the original program. The browser is developed based on the CERN libwww version 2.0. CERN libwww 2.0 is a uniform library of common code developed as the basis for building World Wide Web clients and servers. It contains code for accessing HTTP, FTP, Gopher, NNTP and WAIS servers, perform telnet sessions and access the local file system. Furthermore, it provides functionality for loading, parsing and caching graphic objects plus a wide spectrum of generic programming utilities. The other part of the browser is the interface part which is responsible for establishing the access window for users. This part is developed based on Motif which is an event-driven environment. The whole browser program is an event loop, waiting for events to happen until an exit event is triggered. Once any event is triggered, the corresponding event handler will be called to process the request and it will cooperates with both the library part and the interface part.

A.1 Data Structure

Two header files are mainly used by Mosaic, they are : mosaic.h and xresources.h.
(a) mosaic.h: The main content of mosaic.h is related to information needed by a window, as well as all parameters related to the menubar specification. The global variables are also defined here.

1. `mo_window`: This is the basic and most important data structure of the whole program, it is a structure with all definitions of the information related to the 'Document View Window', such as menubar, buttons, accept_fonts, accept_languages, etc.
2. `mo_node`: Every 'Document View Window' can be considered as a node. Users can switch from one node to another through clicking 'back' or 'forward' buttons on the window. The whole trace which one user invokes Mosaic until she/he exits from the program consists of a history list. This structure records every node's position of the history list and the content of the node, including 'url', 'title', 'pointer to text' and so on, it contains pointers pointing to the previous and the next node as well. Thus, a history list is a two-directions list, every item of the list is a mo_node type variable. Figure A.1 illustrates the history list.

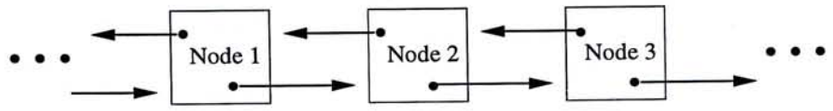

Figure A.1: History List

3. `AppData`: This is a struct too. It comprises all information about the application during its run time. Some of these information can be set by individule user. For example, user can set the following information: use proxy server or not, what is the fontset of the local systems, etc.
4. `mo_token`: This is an enumerate type structure, it defines every concrete value of the menubar, for instance, if a user sets codeset to "gb2312", then the value of mo_token type variable is mo_charset_gb.

(b) xresources.h: This header file contains all default values' definition related to XWindow environment, as well as other resources value. Such as: Font_size, Header FontSet etc; This file also has information on display color of the local system. Each user can replace this file by placing his/her own resource file in a new directory .app-defaults under his home directory.

(c) Global Variables:

1. `mo_window *win;`
   Variable `win` goes through all the corners of the program. The changing of the running program is realized by setting different values of the subitems of variable `win`, including setting menubar, text window, etc.
2. `AppData *Rdata;` Variable `Rdata` nearly plays the same role as `win` does. It records the changing or modification of every important parameters, then the changing of this variable eventually cause the operation change of the whole program.

A.2 Calling Sequence of Functions

The program begins with the main() function which is in the source file *main.c*. The main function of main() is to do the initialization and call another kernel module *mo_do_gui()*. The modules invoked by *mo_do_gui()* are listed as follows:

```
1. mo_setup_global_history()
2. mo_init_global_histort()
3. mo_setup_default_hoslist()
4. mo_write_default_hotlist()
5. mo_setup_pan_list()
6. mo_open_initial_window()
```

The first five functions are called to set up the necessary resources for users to save or access in the future. The sixth function *mo_open_initial_window()* is invoked to establish the first window. The event loop is formed in function *mo_do_gui()* after module *mo_open_initial_window()* is completed.

After function *mo_open_initial_window()* is invoked, it continues to call other functions to complete the task:

1. `mo_make_window()`
2. `mo_set_current_cached_win()`
3. `mo_load_window_text()`

Module *mo_make_window()* makes a new window by setting up the window structure and fill up GUI (Graphic User Interface). This module is related to the browser interface part which has been modified to support the internationalization and localization concepts. Module *mo_set_current_cached_win()* saves the current window information into cache. Module *mo_load_window_text()* is the core function to do the real document access. It tries to establish the HTTP connection with the remote server, and collect data coming from the network channel, and then process the response message and invokes the corresponding event handlers to do the job. This is the module which utilizes the routines of the libwww2.0 which has been modified to support the codeset announcement between web servers and web browsers.

The main function used for establishing a new window is module *mo_fill_window()*. It takes a new(empty) *mo_window* structure and fills in all the GUI elements, including icons, menubars, buttons and so on. Module *mo_fill_window()* calls function *mo_make_document_view_menubar()* to complete most of the works. Function *mo_grok_menubar()* establishes the menubar with the help of some static data structures which are the templates of different kinds of menubars. After establishing the menubar, some event handlers are also needed to make responses when any event is triggered. This can be realized through calling function *XmxCallback(menubar_cb)*. The important events that are triggered through the menubar are listed below:

Button Name	Event ID	CallBack Function
New Window	mo_new_window	mo_open_another_window
Clone Window	mo_clone_window	mo_duplicate_window
Open URL	mo_open_document	mo_post_open_window
Open Local	mo_local_document	mo_post_open_local_window
Reload Current	mo_reload_document	mo_reload_window_text
Reload Images	mo_reload_document _and_images	mo_reload_window_text
Refresh Current	mo_refresh_document	mo_refresh_window_text
Find in Current	mo_search	mo_post_search_window
View Source	mo_document_source	mo_post_source_window
Save As	mo_save_document	mo_post_save_window
Print	mo_print_document	mo_post_print_window
Mail To	mo_mail_document	mo_post_mail_window
Close Window	mo_close_window	mo_delete_window
Exit Program	mo_exit_program	mo_post_exitbox

Another important event clusters are for fonts, different fonts (codesets) have different event handlers respectively. For example, if users choose font GB2312 in the menubar option, then event identification number *mo_charset_gb* is used to invoke the related event handler. In this case, function *mo_force_font_change(win, i)* is called to fulfill users' requirement.

As we see in the above event list, event *Open URL* is a typical event which requires the HTTP connection with the remote server. It invokes function *mo_pull_er_over()* to access the

document with the URL sent from the menubar. Using the given URL information, function *HTLoadAbsolute(url)* is invoked to establish the HTTP connection, sending out HTTP request message, receiving the response message and then do the parsing. Then the body data of the response message will be transferred to the next function *hack_htmlsrc()* for display. This function analyze the body data according to the HTML specification and tries to display them on the browser screen at last.

A.3 Modification of Souce Code

The modification of the source code can be divided three parts: the header file, the interface part and the library part.

Header Files
There are two header files mosaic.h and xresources.h which had been modified. One more new header file my.h is added into the system.

```
1. mosaic.h
mo_window:
            /* Yao Jian's modification */
            char accept_charsets[30];

Note: this variable is used to send codeset information supported
      by the web browser.

2. xresources.h

/* Yao Jian's modification */
/*
"*TitleFont: -adobe-times-bold-r-normal-*-24-*-*-*-*-*-iso8859-1",
*/
*TitleFont: hanzigb16st",

Note: change the font to be simplified Chinese.

/* Yao Jian's modification */
"*defaultCharset:            iso-8859-1",

Note: change the default font set to be iso-8859-1.

3. New header file my.h

/*************************************************************
 *  This header file is used by codeconv.c and gui-menubar.c    *
 *  It defines a struct type used by modified gui-menubar.c     *
 *                                                              *
 *  Yao Jian,  Nov. 12, 1996                                    *
 *************************************************************/

#define MY_BUF  50
```

```
typedef struct my_codeset{
                    int  font_size;  /* related to win->font_size */
                    char language[5];
                    char codeset_name[20];
} MY_CODESET;

Note: define a new data structure to keep the codeset information.
      This is used for the mime parsing later.
```

Interface Part

Some files related to the browser interface has been modified to fulfill the special requirements of handling Chinese.

```
1. gui.c

/* Add By Yao Jian: Apr. 29, 1997 */
#include <locale.h>
#include <nl_types.h>
/* End of Yao Jian's modification */

Note: header files for internationalization and message catalog
      system.

static mo_status mo_fill_window (mo_window *win)
{
   /* Add by Yao Jian on Apr. 26, 1997 */

   nl_catd catd;        /* catd is defined in main.c */

   /* End of Yao Jian's modification */
  ... ...
}

Note: variable catd is used for open a message catalog file.

In module mo_fill_window():

    /* Add by Yao Jian: Apr. 29, 1997 */
    catd = catopen("Mosaicnew.cat",0);

    if ( catd == (nl_catd) -1 ) {
         printf("Open message catalog fails!\n");
         exit(-1);
    }

Note: to open a message catalog file and check error.

In module mo_fill_window():

    /* Use the Message catalog: Yao jian Apr. 30, 1997 */
```

```
    win->back_button = XmxMakeNamedPushButton
        (win->button_rc, catgets(catd, 1, 7, "Back"), "Back", menubar_cb,
         mo_back);

      /* Use message catalog file: Yao Jian Apr. 30, 1997 */
      win->forward_button = XmxMakeNamedPushButton(win->button_rc,
catgets(catd, 1, 8, "Forward"), "Forward", menubar_cb, mo_forward);

      /* Use message catalog file Yao Jian Apr. 30 1997 */
      win->home_button = XmxMakeNamedPushButton(win->button_rc,
catgets(catd, 1, 9, "Home"), "Home", menubar_cb, mo_home_document);

     /* Use message catalog file: Yao Jian Apr. 30, 1997 */
     win->reload_button = XmxMakeNamedPushButton(win->button_rc,
catgets(catd, 1, 10, "Reload"), "Reload", menubar_cb,
mo_reload_document);

      /* Use message catalog file: Yao Jian Apr. 30, 1997 */
      win->open_button =  XmxMakeNamedPushButton(win->button_rc,
catgets(catd, 1, 11, "Open"), "Open", menubar_cb, mo_open_document);

   /* Use message catalog file: Yao Jian Apr. 30, 1997 */
   win->save_button = XmxMakeNamedPushButton(win->button_rc,
catgets(catd, 1, 12, "Save As..."), "SaveAs", menubar_cb,
mo_save_document);

        /* Use message catalog file on Apr. 30, 1997 */
        win->clone_button = XmxMakeNamedPushButton(win->button_rc,
catgets(catd, 1, 13, "Clone"), "Clone", menubar_cb, mo_clone_window);

      /* Use message catalog file: Yao Jian Apr. 30, 1997 */
          win->new_button = XmxMakeNamedPushButton(win->button_rc,
catgets(catd, 1, 14, "New Window"), "NewWindow", menubar_cb,
mo_new_window);

      /* Use message catalog file: Yao Jian Apr. 30, 1997 */
        win->close_button = XmxMakeNamedPushButton(win->button_rc,
catgets(catd, 1, 15, "Close Window"), "CloseWindow", menubar_cb,
mo_close_window);
```

Note: Use messages in the message catalog file instead of hardcoding them in the program.

In module mo_make_window():

```
  /* Yao Jian's modification */
  XmxSetArg (XmNtitle, (long)"CUHK : Document View ");
```

Note: change the title of the browser.

2. gui-dialogs.c

```
/* Yao Jian's modification */

char *mo_get_accept_charsets()
{
  return(current_win->accept_charsets);
}
```

Note: this is the new defined function to get the codeset information.
this codeset information will be sent to the remote server
after the HTTP connection is established.

3. gui-documents.c

In module mo_do_window_text():

```
    /* Yao Jian's modification: to set font_size through checking url */
    /* Just remove it, then see what happen?  Oct. 22, 0:30 am   */
    /*
    {  char site[50];
       int  my_length=0;

       strcpy(site, url);
       my_length=strlen(site);
       if ( (site[my_length-1] == 'b') && (site[my_length-2] == 'g') )
          mo_force_font_change(win, mo_charset_gb);
       if ( (site[my_length-1] == '5') && (site[my_length-2] == 'b') )
          mo_force_font_change(win, mo_charset_big5);
    }
    */
    /* end of Yao Jian's modification */
```

Note: to check the codeset of the document through investigating
the file name extension.

In module mo_ml_codeconv():

```
    /* Yao Jian's modification: both c1 and c2 are EUC code */
    if (eucflag == 1)
    {
        if ( (0xb0 <= c1) && (c1 <= 0xf7) )
             mo_set_fonts(win, mo_charset_gb);
        }
    /* end of Yao Jian's modification */
```

Note: to check the codeset by investigating the binary code.

4. gui-menubar.c

```
/* Yao Jian's modification */
```

```
#include "my.h"
#include "HText.h"

/* Yao Jian's modification on Apr. 26, 1997 */
#include <locale.h>
#include <nl_types.h>

/* Add by Yao Jian on Apr. 26, 1997 */

nl_catd catd;       /* catd is defined in main.c */

/* End of Yao Jian's modification */

/* Yao Jian's modification */

/* extern HText* HTMainText;*/ /* a global variable for storing text */

extern char AfterMimeCodeset[30]; /* keep codeset after mime parsing */

/* codeset conversion function */
extern void conv_str(char *sourcetext, char *desttext, char *fromcode,
                     char *tocode, int textlen);

extern MY_CODESET MyFont[13];  /* save font information */
                               /* MyFont is a global variable */

extern int to_init;     /* decide to init MyFont or not */

/* Yao Jian's modification on Apr. 26, 1997 */

nl_catd catd;   /* for opening a message catalog file */


/* Yao Jian's modification on Apr. 26, 1997 */

nl_catd catd;   /* for opening a message catalog file */

extern void HText_clearOutForNewContents(HText *self);

extern void HText_appendBlock(HText *text, char *data, int len);

struct _HText {
  char *expandedAddress;
  char *simpleAddress;

  /* This is what we should parse and display; it is *not*
     safe to free. */
  char *htmlSrc;
  /* This is what we should free. */
  char *htmlSrcHead;
```

```
  int srcalloc;    /* amount of space allocated */
  int srclen;      /* amount of space used */
};
/* End of Yao Jian's modification */
```

Note: add some header files for supporting internationalization and the new header file is for automatic codeset conversion.

In module mo_set_fonts():

```
      /* Yao Jian's modification */
      strcpy(win->accept_charsets, "iso-8859-1");
```

Note: add acceptable codeset information for all codesets.

In module mo_set_fonts():

```
      if (size == mo_charset_gb)
      {
       XmxSetArg (WbNcharsetInfo, CS_GB);
       /* Yao Jian's modification */
       /*
       strcpy(win->accept_charsets, "iso-8859-1, gb2312");
       */
       strcpy(win->accept_charsets, "gb");

      /* Yao jian's modification */
      /*
      strcpy(win->accept_charsets, "iso-8859-1, big5");
      */
      strcpy(win->accept_charsets, "Big5");
```

Note: add acceptable codeset information for gb2312 and Big5.

In module mo_force_font_change():

```
  /* Yao Jian's definition */

  MY_CODESET pre, back;
  char *p;          /* pointing to the source text */
  char *temp;       /* store converted text */

  int  i;
  int  outlen = 0;

  /* End of Yao Jian's definition */
```

Note: define related variables for automatic codeset conversion.

In module mo_force_font_change():

```
  /* Yao Jian's modification */

  /* initialize pre and back */
  pre.font_size = win->current_node->font_size; /* save previous font */
  pre.language[0] = '\0';
  pre.codeset_name[0] = '\0';

  back.font_size=size;  /* save new font */
  back.language[0] = '\0';
  back.codeset_name[0] = '\0';

  /* End of Yao Jian's modification */
```

Note: to keep the original codeset setting and the latest codeset setting.

In module mo_force_font_change():

```
  /* Yao Jian's modification */

  if (to_init == 0 ) {
    to_init = 1;
    font_init();
  }

  /* get information before font setting */
  for (i=0; i<13; i++) {
     if(pre.font_size == MyFont[i].font_size)
     {
        strcpy(pre.language, MyFont[i].language);
        strcpy(pre.codeset_name, MyFont[i].codeset_name);
        break;
     }
  } /* end of for */

  /* get information after font setting */
  for (i=0; i<13; i++) {
     if(back.font_size == MyFont[i].font_size)
     {
        strcpy(back.language, MyFont[i].language);
        strcpy(back.codeset_name, MyFont[i].codeset_name);
        break;
     }
  } /* end of for */

  /* add codeset conversion hereafter */
  if ( (strcmp(pre.language,back.language) == 0) &&
       (strcmp(pre.codeset_name,back.codeset_name) !=0) &&
       (strcmp(pre.language,"zh") == 0) )  {
```

```
   outlen = strlen(win->current_node->text);
   p = win->current_node->text;
 /* allocate memory to temp */
 temp = (char *) malloc (outlen * sizeof(char) + 200 );
 if (temp == NULL) {
    printf("not enough memory for temp!\n");
    exit(-1);
 }
 else
    temp[0] = '\0';  /* empty it first */

   conv_str(p, temp, pre.codeset_name, back.codeset_name, outlen);

   /* Force program to redisplay the window, Nov. 5, 1996 */

   /* clear out the old contents of main text */
   HText_clearOutForNewContents(HTMainText);

   /* append new contents to main text */

   HText_appendBlock(HTMainText, temp, strlen(temp) );

   /* make pointers pointing correctly */
   win->current_node->texthead = HTMainText->htmlSrc;
   win->current_node->text = HTMainText->htmlSrc;

   mo_set_win_current_node(win, win->current_node);
   free(temp);

  } /* end of if */

  /* End of Yao Jian's modification */
```

Note: add codeset conversion here.

Add a new function ingetmessage() in source file gui-menubar.c:

```
/* Add by Yao Jian on Apr. 28, 1997 */
void  ingetmessage()
{

  catd = catopen("Mosaicnew.cat",0);

  if ( catd == (nl_catd) -1 ) {
     printf("Open message catalog fails!\n");
     exit(-1);
  }

  /* modify structure menuspec */
  menuspec[0].namestr = catgets(catd, 1, 1, "Yao Jian");
  menuspec[1].namestr = catgets(catd, 1, 2, "Yao Jian");
```

```
menuspec[2].namestr = catgets(catd, 1, 3, "Yao Jian");
menuspec[3].namestr = catgets(catd, 1, 4, "Yao Jian");
menuspec[4].namestr = catgets(catd, 1, 5, "Yao Jian");
menuspec[5].namestr = catgets(catd, 1, 6, "Yao Jian");

/* modify structure file_menuspec */
file_menuspec[0].namestr = catgets(catd, 2, 1, "Li Xiang Ping");
file_menuspec[1].namestr = catgets(catd, 2, 2, "Li Xiang Ping");
file_menuspec[3].namestr = catgets(catd, 2, 3, "Li Xiang Ping");
file_menuspec[4].namestr = catgets(catd, 2, 4, "Li Xiang Ping");
file_menuspec[6].namestr = catgets(catd, 2, 5, "Li Xiang Ping");
file_menuspec[7].namestr = catgets(catd, 2, 6, "Li Xiang Ping");
file_menuspec[8].namestr = catgets(catd, 2, 7, "Li Xiang Ping");
file_menuspec[10].namestr = catgets(catd, 2, 8, "Li Xiang Ping");
file_menuspec[11].namestr = catgets(catd, 2, 9, "Li Xiang Ping");
file_menuspec[12].namestr = catgets(catd, 2, 10, "Li Xiang Ping");
file_menuspec[14].namestr = catgets(catd, 2, 11, "Li Xiang Ping");
file_menuspec[15].namestr = catgets(catd, 2, 12, "Li Xiang Ping");
file_menuspec[16].namestr = catgets(catd, 2, 13, "Li Xiang Ping");
file_menuspec[18].namestr = catgets(catd, 2, 14, "Li Xiang Ping");
file_menuspec[20].namestr = catgets(catd, 2, 15, "Li Xiang Ping");
file_menuspec[21].namestr = catgets(catd, 2, 16, "Li Xiang Ping");

/* modify structure opts_menuspec */
opts_menuspec[0].namestr = catgets(catd, 2, 17, "Yao Peng");
opts_menuspec[2].namestr = catgets(catd, 2, 18, "Yao Peng");
opts_menuspec[4].namestr = catgets(catd, 2, 19, "Yao Peng");
opts_menuspec[5].namestr = catgets(catd, 2, 20, "Yao Peng");
opts_menuspec[7].namestr = catgets(catd, 2, 21, "Yao Peng");
opts_menuspec[9].namestr = catgets(catd, 2, 22, "Yao Peng");
opts_menuspec[10].namestr = catgets(catd, 2, 23, "Yao Peng");
opts_menuspec[12].namestr = catgets(catd, 2, 24, "Yao Peng");
opts_menuspec[13].namestr = catgets(catd, 2, 25, "Yao Peng");
opts_menuspec[14].namestr = catgets(catd, 2, 26, "Yao Peng");
opts_menuspec[15].namestr = catgets(catd, 2, 27, "Yao Peng");
opts_menuspec[17].namestr = catgets(catd, 2, 28, "Yao Peng");

/* modify structure navi_menuspec */
navi_menuspec[0].namestr = catgets(catd, 2, 29, "Jian");
navi_menuspec[1].namestr = catgets(catd, 2, 30, "Jian");
navi_menuspec[3].namestr = catgets(catd, 2, 31, "Jian");
navi_menuspec[4].namestr = catgets(catd, 2, 32, "Jian");
navi_menuspec[6].namestr = catgets(catd, 2, 33, "Jian");
navi_menuspec[7].namestr = catgets(catd, 2, 34, "Jian");
navi_menuspec[9].namestr = catgets(catd, 2, 35, "Jian");
navi_menuspec[10].namestr = catgets(catd, 2, 36, "Jian");

/* modify structure anno_menuspec */
anno_menuspec[0].namestr = catgets(catd, 2, 37, "Ping");
anno_menuspec[1].namestr = catgets(catd, 2, 38, "Ping");
anno_menuspec[3].namestr = catgets(catd, 2, 39, "Ping");
```

```
anno_menuspec[4].namestr = catgets(catd, 2, 40, "Ping");

/* modify structure news_menuspec */
news_menuspec[0].namestr = catgets(catd, 2, 41, "Peng");
news_menuspec[1].namestr = catgets(catd, 2, 42, "Peng");
news_menuspec[2].namestr = catgets(catd, 2, 43, "Peng");
news_menuspec[3].namestr = catgets(catd, 2, 44, "Peng");
news_menuspec[4].namestr = catgets(catd, 2, 45, "Peng");
news_menuspec[5].namestr = catgets(catd, 2, 46, "Peng");
news_menuspec[7].namestr = catgets(catd, 2, 47, "Peng");
news_menuspec[8].namestr = catgets(catd, 2, 48, "Peng");
news_menuspec[10].namestr = catgets(catd, 2, 49, "Peng");

/* modify structure help_menuspec */
help_menuspec[0].namestr = catgets(catd, 2, 50, "Peng");
help_menuspec[1].namestr = catgets(catd, 2, 51, "Peng");
help_menuspec[3].namestr = catgets(catd, 2, 52, "Peng");
help_menuspec[4].namestr = catgets(catd, 2, 53, "Peng");
help_menuspec[6].namestr = catgets(catd, 2, 54, "Peng");
help_menuspec[7].namestr = catgets(catd, 2, 55, "Peng");
help_menuspec[8].namestr = catgets(catd, 2, 56, "Peng");
help_menuspec[10].namestr = catgets(catd, 2, 57, "Peng");
help_menuspec[11].namestr = catgets(catd, 2, 58, "Peng");
help_menuspec[13].namestr = catgets(catd, 2, 59, "Peng");
help_menuspec[15].namestr = catgets(catd, 2, 60, "Peng");

/* modify structure fnts_menuspec */
fnts_menuspec[0].namestr = catgets(catd, 3, 1, "Yao");
fnts_menuspec[1].namestr = catgets(catd, 3, 2, "Yao");
fnts_menuspec[2].namestr = catgets(catd, 3, 3, "Yao");
fnts_menuspec[4].namestr = catgets(catd, 3, 4, "Yao");
fnts_menuspec[5].namestr = catgets(catd, 3, 5, "Yao");
fnts_menuspec[6].namestr = catgets(catd, 3, 6, "Yao");
fnts_menuspec[8].namestr = catgets(catd, 3, 7, "Yao");
fnts_menuspec[9].namestr = catgets(catd, 3, 8, "Yao");
fnts_menuspec[10].namestr = catgets(catd, 3, 9, "Yao");
fnts_menuspec[12].namestr = catgets(catd, 3, 10, "Yao");
fnts_menuspec[13].namestr = catgets(catd, 3, 11, "Yao");
fnts_menuspec[14].namestr = catgets(catd, 3, 12, "Yao");
fnts_menuspec[16].namestr = catgets(catd, 3, 13, "Yao");
fnts_menuspec[17].namestr = catgets(catd, 3, 14, "Yao");
fnts_menuspec[18].namestr = catgets(catd, 3, 15, "Yao");
fnts_menuspec[19].namestr = catgets(catd, 3, 16, "Yao");
fnts_menuspec[20].namestr = catgets(catd, 3, 17, "Yao");
fnts_menuspec[21].namestr = catgets(catd, 3, 18, "Yao");
fnts_menuspec[22].namestr = catgets(catd, 3, 19, "Yao");
fnts_menuspec[23].namestr = catgets(catd, 3, 20, "Yao");
fnts_menuspec[24].namestr = catgets(catd, 3, 21, "Yao");
fnts_menuspec[25].namestr = catgets(catd, 3, 22, "Yao");
fnts_menuspec[26].namestr = catgets(catd, 3, 23, "Yao");
fnts_menuspec[27].namestr = catgets(catd, 3, 24, "Yao");
```

```
  fnts_menuspec[28].namestr = catgets(catd, 3, 25, "Yao");

  /* modify structure bidir_menuspec */
  bidir_menuspec[0].namestr = catgets(catd, 3, 26, "Zhang");
  bidir_menuspec[1].namestr = catgets(catd, 3, 27, "Zhang");

  /* modify structure undr_menuspec */
  undr_menuspec[0].namestr = catgets(catd, 3, 28, "Zhang");
  undr_menuspec[1].namestr = catgets(catd, 3, 29, "Zhang");
  undr_menuspec[2].namestr = catgets(catd, 3, 30, "Zhang");
  undr_menuspec[3].namestr = catgets(catd, 3, 31, "Zhang");
  undr_menuspec[4].namestr = catgets(catd, 3, 32, "Zhang");

  /* modify structure newsfmt_menuspec */
  newsfmt_menuspec[0].namestr = catgets(catd, 3, 33, "Zhang");
  newsfmt_menuspec[1].namestr = catgets(catd, 3, 34, "Zhang");
}

/* End of Yao Jian's modification on Apr. 28, 1997 */

Note: This new function plug program messages from the message
      catalog files.

In module mo_make_document_view_menubar():

  /* Add by Yao Jian on Apr. 28, 1997 */

  ingetmessage();

  /* End of Modification by Yao Jian */

Note: to get message from the message catalog files.

5. codeconv.c
This is a new source file used to do the automatic codeset conversion.
This file contains two modules: conv_str() and font_init().
Module conv_str() invokes the codeset conversion routines defined
in the I-Hanzix server, and it handles every exception cases.

/*******************************************************
 * Name: codeconv.c                                    *
 * Function: codeset conversion as a function          *
 * API: void conv_str(char *sourcetext, *desttext,     *
 *                    char *fromcode, char *tocode,    *
 *                    int  textlen)                    *
 * Yao Jian, Nov. 12, 1996                             *
 *******************************************************/
#include "my.h"
#include "stdlib.h"
#include "stdio.h"
#include  <ctype.h>
```

```
#include  <memory.h>
#include  <fcntl.h>
#include  <sys/types.h>
#include  <sys/stat.h>
#include  "ihanzix.h"
#include  "hzconverter.h"

MY_CODESET MyFont[13];  /* save font information, a global variable */

int  to_init = 0;     /* decide to init MyFont or not */

extern CONVERSION_INFO IHanzixGetConversionInfo(char *fromcode, char
*tocode);

int ShadowErrorNo;

static void shift_bytes(char *from, char *to, unsigned int nbytes);

void conv_str(char *sourcetext, char *desttext, char *fromcode,
              char *tocode, int textlen)
{
   char *p, *q;
   int  inlen=0;
   char inbuf[MY_BUF], outbuf[MY_BUF*4];
   char *inbufptr = inbuf, *outbufptr = outbuf;
   unsigned int inbytesleft = 0,
                outbytesleft = sizeof(outbuf);

   const unsigned int insize = sizeof(inbuf), outsize = sizeof(outbuf);
   int nchars = 0, nbytes;

   CONVERSION_INFO custom;
   size_t result;

   hz_iconv_p  iconv_fp = NULL;

   inbuf[0]='\0';
   outbuf[0]='\0';

   p = sourcetext;

   if (iconv_fp == NULL) {
       inbufptr = inbuf; outbufptr = outbuf;

       if ((iconv_fp = hz_iconv_open(tocode, fromcode, NULL)) == NULL) {
        fprintf(stderr, "%s: cannot open the required converter\n");
         exit(-1);
       }
   }

   if (( custom = IHanzixGetConversionInfo(fromcode, tocode)) == NULL )
```

```
{
      printf("cannot get information for custom!\n");
      exit(-3);
}

while( p < sourcetext + textlen ) {

  q = inbufptr;
  while ( *p != '\0' && q < inbufptr+ (insize-inbytesleft) ) {
     *q = *p;
     q++;
     p++;
  }

  nbytes = q - inbufptr;
  /* reset the pointer to point to beginning of buffer */
  inbufptr = inbuf;

  inbytesleft += nbytes;

  result = hz_iconv(iconv_fp, &inbufptr, &inbytesleft,
                    &outbufptr, &outbytesleft,
                    NULL, &nchars, custom);

  switch ( result ) {
    case HZ_ICONV_OK:
    case HZ_ICONV_DONE:

       /* printf("outbuf is %s\n", outbuf); */
       strncat(desttext, outbuf, outsize - outbytesleft);
       inbufptr = inbuf; outbufptr = outbuf;
       inbytesleft = 0; outbytesleft = outsize;
       break;

    case HZ_ICONV_OVER :     /* output buffer is full */

       strncat(desttext, outbuf, outsize - outbytesleft);
       outbufptr = outbuf; outbytesleft = outsize;
       shift_bytes(inbufptr, inbuf, inbytesleft);
       inbufptr = inbuf;
       break;

    case HZ_ICONV_INVAL : /* invalid character detected in input */

       /* output the converted chars, if any */
       strncat(desttext, outbuf, outsize - outbytesleft);
       outbufptr = outbuf; outbytesleft = outsize;

       /* removing 2 bytes and retry again */
       printf("invalid char found at byte offset %d\n",
              inbufptr - inbuf);
```

```
          inbufptr += 2; inbytesleft -= 2;

          /* have we gone beyond the input buffer size ? */

          if (inbufptr - inbuf > insize) {
            /* if so, reset the pointer */
            inbufptr = inbuf; inbytesleft = 0;
          }
          else {
          /* move the unconverted bytes to the beginning of the input
buf */
            shift_bytes(inbufptr, inbuf, inbytesleft);

            /* the next read should start here */
            inbufptr = inbuf + inbytesleft;
          }
          break;

        case HZ_ICONV_TRUNC :   /* partial char is detected */

          strncat(desttext, outbuf, outsize - outbytesleft);
          shift_bytes(inbufptr, inbuf, inbytesleft);
          inbufptr = inbuf; inbufptr = inbuf + inbytesleft;
          outbufptr = outbuf; outbytesleft = outsize;
          break;

        case HZ_ICONV_ENOMEM :  /* not possible, we don't want
exceptions */
        default :
          fprintf(stderr, "%s : internal error of hz_iconv() ??\n");


          exit(-99);
     } /* end of switch */

   } /* end of while */

   hz_iconv_close(iconv_fp); /* close the converter */

}

static void shift_bytes(char *from, char *to, unsigned int nbytes)
{
    memcpy((void *) to, (void *) from, nbytes);
}

void font_init()
{

    /* save value into MyFont[13] first */
```

```
MyFont[0].font_size=42;
strcpy(MyFont[0].language,"en");  /* English */
strcpy(MyFont[0].codeset_name,"ISO-8859-2");

MyFont[1].font_size=43;
strcpy(MyFont[1].language,"en");  /* English */
strcpy(MyFont[1].codeset_name,"ISO-8859-3");

MyFont[2].font_size=44;
strcpy(MyFont[2].language,"en");  /* English */
strcpy(MyFont[2].codeset_name,"ISO-8859-4");

MyFont[3].font_size=45;
strcpy(MyFont[3].language,"en");  /* English */
strcpy(MyFont[3].codeset_name,"ISO-8859-5");

MyFont[4].font_size=46;
strcpy(MyFont[4].language,"en");  /* English */
strcpy(MyFont[4].codeset_name,"ISO-8859-7");

MyFont[5].font_size=47;
strcpy(MyFont[5].language,"en");  /* English */
strcpy(MyFont[5].codeset_name,"ISO-8859-8");

MyFont[6].font_size=48;
strcpy(MyFont[6].language,"en");  /* English */
strcpy(MyFont[6].codeset_name,"ISO-8859-9");

MyFont[7].font_size=49;
strcpy(MyFont[7].language,"");  /* Empty */
strcpy(MyFont[7].codeset_name,"KOI8");

MyFont[8].font_size=50;
strcpy(MyFont[8].language,"zh");  /* Chinese */
strcpy(MyFont[8].codeset_name,"GB");

MyFont[9].font_size=51;
strcpy(MyFont[9].language,"zh");  /* Chinese */
strcpy(MyFont[9].codeset_name,"HZ");

MyFont[10].font_size=52;
strcpy(MyFont[10].language,"jp");  /* Japanese */
strcpy(MyFont[10].codeset_name,"JIS");

MyFont[11].font_size=53;
strcpy(MyFont[11].language,"kr");  /* Korean */
strcpy(MyFont[11].codeset_name,"KSC");

MyFont[12].font_size=54;
strcpy(MyFont[12].language,"zh");  /* Chinese */
strcpy(MyFont[12].codeset_name,"BIG5");
```

```
}
```

Library Part

The modification of the library part is related mainly to the HTTP connection part. There are two source files which have been modified. They are HTTP.c and HTMIME.c.

```
1. HTTP.c
This is the file containing all related functions which carry
out the work to establish the HTTP connection with
the remote server.

/* Yao Jian's modification */

extern char * mo_get_accept_charsets();

Note: declare the new function which is defined in another
      source file.

In module HTLoadHTTP():

      /* Yao Jian's modification */
      { char *mystr;
           /* strcpy(mystr, mo_get_accept_charsets()); */
           if ((mystr = strdup(mo_get_accept_charsets())) != NULL)
           {
               printf("mystr is: %s\n", mystr);
               sprintf(line, "Accept-Charset: %s%c%c", mystr, CR, LF);
               StrAllocCat(command, line);
           }

      }

Note: add acceptable codeset information into the HTTP request
      message.

2. HTMIME.c
This is the file which contains the functions for parsing
the response message from the remote server.

/* Yao Jian's modification */

char AfterMimeCodeset[30]="";  /* This is a global variable     */
                               /* Other functions can use it as */
                               /* an extern variable            */

Note: define a structure for codeset conversion.

In module HTMIME_put_character():

        case '\n':                        /* Blank line: End of Header! */
```

```
        {
            int compressed = COMPRESSED_NOT;

            /* Yao Jian's modification */
            {  char *found1, *found2, *head;
               char temp[30];
               int  len, sub_len;
               char target_codeset[30];

               head=me->format->name;    /* point to the head */
               found1=strstr(head, ";");
               found2=strstr(head, "charset");
               if(found2)  /* have charset */
               {
                   sub_len=found1-head;
                   strncpy(temp, head, sub_len); /*temp:first part */
                   temp[sub_len]='\0';
                   found1=strstr(found2, "=");
                   if(found1)
                   {
                     len=strlen(found2);
                     sub_len=len-(found1-found2);
                     found1++;
                     strcpy(target_codeset, found1);
                     /* Yao Jian: save it into global */
                     strcpy(AfterMimeCodeset, target_codeset);
                   }
                 /* Yao: should modify structure instead of string */

                   /*
                   strcpy(me->format->name, temp);
                   */
                   me->format = HTAtom_for(temp);
                   printf("format name is: %s\n", me->format->name);
                   printf("target codeset is: %s\n", target_codeset);
                   printf("AfterMimeCodeset is: %s\n",
AfterMimeCodeset);
                 }
             }
             /* end of Yao Jian's modification */

Note: add the capability to analyze the charset parameter in the
      HTTP response message header field Content-Type.

3. Xmx.h

In Xmx.h:

/* Menubar uses a recursive struct. */
typedef struct _XmxMenubarStruct
{
```

```
  /* Modified by Yao Jian on Apr. 28, 1997 */
  String namestr;
  /* comment by Yao Jian on June 12, 1997 */
  /*
  char namestr[60];
  */
  char mnemonic;
  void (*func)();
  int data;
  struct _XmxMenubarStruct *sub_menu;
} XmxMenubarStruct;

Note: modify the string pointer to be a string array,
      so that Chinese characters can be saved into
      the structure without space problems.
```

A.4 Modification of Resources

The resource files for different codesets have been modified to support the right fonts and right codesets. All resources files are located under /local/ciswb/.app-defaults/ directory, there are different locales for different codesets: zh for gb2312, zh_TW for CNS 11643, BIG5 for Big5, en_US for English. Under each locale, the resource file has the same name: Mosaic, its content is different for various codesets.

```
1. GB 2312 resource file:

Mosaic*XmLabel*fontList:                8x16;hanzigb16st:
Mosaic*XmPushButton*fontList:           8x16;hanzigb16st:
Mosaic*XmToggleButton*fontList:         8x16;hanzigb16st:
Mosaic*optionmenu*fontList:             8x16;hanzigb16st:
Mosaic*menubar*fontList:                8x16;hanzigb16st:
Mosaic*pulldownmenu*fontList:           8x16;hanzigb16st:

Mosaic*defaultCharset:                  gb

Note: modify the font to support simplified Chinese.

2. Big5 resource file:

Mosaic*XmLabel*fontList: -*-*-medium-r-normal-*-16-160-72-72-c-160-big5*-*
Mosaic*XmPushButton*fontList: -*-*-medium-r-normal-*-16-160-72-72-c-160-big5*-*
Mosaic*XmToggleButton*fontList: -*-*-medium-r-normal-*-16-160-72-72-c-160-big5*-*
Mosaic*optionmenu*fontList: -*-*-medium-r-normal-*-16-160-72-72-c-160-big5*-*
Mosaic*menubar*fontList: -*-*-medium-r-normal-*-16-160-72-72-c-160-big5*-*
Mosaic*pulldownmenu*fontList: -*-*-medium-r-normal-*-16-160-72-72-c-160-big5*-*

Mosaic*defaultCharset:                  Big5
```

Note: modify the font to support traditional Chinese -- Big5 codeset.

3. CNS 11643 resource file:

```
Mosaic*XmLabel*fontList:                8x16;cns116;cns216:
Mosaic*XmPushButton*fontList:           8x16;cns116;cns216:
Mosaic*XmToggleButton*fontList:         8x16;cns116;cns216:
Mosaic*optionmenu*fontList:             8x16;cns116;cns216:
Mosaic*menubar*fontList:                8x16;cns116;cns216:
Mosaic*pulldownmenu*fontList:           8x16;cns116;cns216:

Mosaic*defaultCharset:                  cns
```

Note: modify the font to support traditional Chinese -- CNS codeset.

Appendix B

User Manual

It is quite straitforward to use our web system to do Chinese information access. The web server is located on machine *http://cweb1:8001/*. Both the web server and the web browser run on the Unix platform.

Before invoking the web browser, it is users' responsibility to set the local environment. It means users have to set the codeset they prefer before invoking the web browser. The commands they can use are:

```
1. setenv LANG zh :   setting to gb2312 (simplified Chinese)
2. setenv LANG BIG5:  setting to Big5   (traditional Chinese)
3. setenv LANG zh_TW: setting to CNS 11643 (traditional Chinese)
4. setenv LANG en_US: setting to English
```

Since the web browser has been internationalized, everytime, before it establishes the first window, all program messages are extracted from related message catalog files. To do so, users have to set up other system environment so that the program can search and fetch the messages and plug them into the program. The commands users can use are:

```
setenv NLSPATH /local/MosaicBAK/message/%L/%N

Note: this is used to set the path for the browser to search for
      the related message catalog file.

setenv LD_LIBRARY_PATH /usr/X11R5/lib:/usr/X11R6/lib:\
                       /usr/local/motif2.0/lib:/usr/ucblib:
                       /local/Hanzix/iconv/dlib

Note: this is used to link to the codeset conversion routines
      defined by the I-Hanzix server.
```

Now, users can run the program with the following command:

```
/local/MosaicBAK/src/Mosaic &
```

After the browser sets up the first window, users can access the URL they want as what they used to do when using Mosaic in the past. The only difference is that in the menubar option - font, there are two more Chinese codesets for users to select: GB 2312 and BIG5. Since users have set the codeset before the browser is invoked, by default, the browser will send that codeset as the acceptable codeset by the browser, so if users access a Chinese file encoded in a codeset which is incompatible with what the browser accepts, automatic codeset conversion will be automatically done at the server side, and the converted document will be sent back without any intervention by users. If users want to change the current document's codeset, they can choose the right font in the menubar option - font, and the on-line codeset conversio will be done by the browser.