CJK Characters Handling on Internet for Thin Clients

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING

OF THE HONG KONG POLYTECHNIC UNIVERSITY

FOR THE DEGREE OF

MASTER OF PHILOSOPHY

By

Lo Chi Wing

The Hong Kong Polytechnic University

2002

i. Abstract

Abstract of thesis entitled "CJK Characters Handling on Internet for Thin Clients" submitted by Lo Chi Wing for the degree of Master of Philosophy at the Hong Kong Polytechnic University in October 2002.

Web pages in Chinese, Japanese and Korean (often called CJK) are now very commonly posted in Asian Web sites. No Web browsers can view CJK characters without the installation of suitable fonts. However, font installation is quite complicated and a lot of memory space is required to store fonts, making it infeasible in the thin-client environments. This thesis presents an Internet-based font display server called Pelutech that is primarily intended for display of CJK characters in image forms without the need for font installation. Pelutech acts as a display proxy to process requests and responses between client machines and a Web server. The system relies on the character-to-image conversion technique at the server side to produce small CJK character images to be sent to client machines. The comprehensive features of Pelutech include: (1) the ability to detect font sets on client machines as well as codesets in which requested Web pages are coded; (2) comprehensive HTML parsing and its unique ability to handle form controls and searching so as to maintain the ability to interact during Web browsing even when characters are converted to images; and (3) the provision of an Internet based input method engine so that users can browse CJK Web pages with text input ability independent of the environment of their Web access devices. The proposed Internet-based solution is highly suitable for thin-client machines having only limited hardware or software support.

ii. Acknowledgements

I would like to express my sincere thanks to my supervisor, Dr. Lu Qin, for her guidance throughout my study. She provided a lot of advice and assistance including continuous consultation, meetings, comments and suggestions on this project and on the thesis preparation. She also allowed a sufficient degree of flexibility for me to manage the project and to solve problems independently. In addition, Dr. Lu not only supervised me on academic matters, but also shared with me a lot life experience on how to be a mature human being and how to succeed in life as a person. She treated me with respect and as a peer. I will treasure this for the rest of life.

I would like to thank Dr. James Liu for being my exam panel chair. Special thanks to Dr. Joseph Kee-Yin Ng, my external examiner, who has given me many suggestions on the revision of thesis and formation of performance analysis. Also, my gratitude to Prof. Kam-Fai Wong, my external examiner, who ardently reviewed my thesis with great details and provided me with useful recommendation for the improvement of my work. He also shared with me his experiences and taught me much valuable knowledge for which I am very thankful.

I would like to thank Kwun-Tak Ng, who helped me a lot on the experiments and the evaluation work. We've also written a few research papers together. Furthermore, I would like to thank all my friends at the Hong Kong Polytechnic University who made my life here meaningful and enjoyable. I would like to thank for their input, encouragement, patience, and support. There have my deepest thanks and appreciation.

Last, but not least, I would like to thank my parents and my friends for their support and encouragement.

iii. Table of Content

iv. List of Figures

v. List of Tables

1. Introduction

Continued globalization and internationalization make the use of the World Wide Web (WWW) grow at a tremendous speed. The Internet has become a global network and continues to spread all over the world. A vast majority of Web sites use Western alphabet-based languages, such as English and French, and the Hypertext Markup Language (HTML) uses the Basic Latin ISO-8859-1 character set by default. The increasing popularity and accessibility of the Internet, however, creates a great demand for Internet information and HTML documents to be written in other languages beyond the Latin-based languages [1]. Web pages in Chinese, Japanese and Korean (often called CJK) are now very commonly used in Asian Web sites, and multi-lingual information sometimes appears in the same Web page.

Moreover, due to the rapid development of electronic devices in recent years, people can surf the Internet at any time and anywhere using a mobile phone or a Personal Digital Assistant (PDA), etc. These mobile devices are usually called thin-client devices because they have limited resources. By limited resources we mean either limited hardware such as memory or limited software such as the lack of some required software. In addition, software support in the thin-client environment is still predominantly English-based because most thin-client devices are developed in Western countries.

In order to read CJK characters on the Internet, CJK related fonts must be installed on the client side. No browsers can view CJK characters without the installation of suitable fonts [2]. However, the font installation procedure is quite complicated. A lot of memory space for storage is also required. In the thin-client environment where memory is very limited, it is not only unsuitable to pre-install fonts which might not always be used but also infeasible due to the limited resources of a thin-client [3]. Therefore, there is a need to investigate some alternative methods to display CJK characters that are platform independent and suitable for thin-clients.

In this thesis, we investigate methods to improve the service of character-to-image conversion techniques in terms of its comprehensiveness and its adequacy for thin-client devices. We present an Internet-based font display server called Pelutech as an alternative to local CJK font support for CJK character display. Pelutech acts as a display server to process requests and responses between client machines and a Web server. The system relies on the character-to-image conversion technique at the server side to produce small CJK character images before they are sent to client browsers.

The comprehensive features of Pelutech include: (1) the ability to detect font sets on client machines as well the codesets in which the requested Web pages are coded; (2) comprehensive HTML parsing and its unique ability to handle form controls and searching so as to maintain the ability to interact during Web browsing even when characters are converted to images; and (3) the provision of an Internet based input method engine so that users can browse CJK Web pages with text input ability independent of the environment of their Web access devices.

Pelutech has two major subsystems: the Conversion Engine and the Input Method Engine. The Conversion Engine analyzes a requested Web page and produces a new Web page with CJK characters converted to embedded images so that clients can be free from any CJK font installation. The Input Method Engine provides an interactive interface for users to input CJK characters in order to facilitate interactive form filling and searching without the need to install any third-party CJK product on the client side. The proposed Internet-based solution is highly suitable for thin-clients having only limited hardware and software support.

The rest of this thesis is organized as follows. Chapter 2 presents the basic concepts and reviews some current work related to CJK character display technology. Chapter 3 discusses the design considerations of a good character-to-image system and gives an overview of the Pelutech system architecture. The design specifications of the components in Pelutech are discussed in Chapter 4 through Chapter 8. The evaluation of the system is discussed in Chapter 9. Chapter 10 gives the conclusions and discusses future work.

2. Basic Concepts and Related Work

2.1 The Need for CJK Reading on Internet

Multilingual support on the Internet is an important topic because people of all languages want to access the Web. Table 2-1 lists the twenty major languages used in the world [4]. It is obvious that nearly one-fourth of the world population use either Chinese, Japanese or Korean (often called CJK) characters [5].

However, reading CJK characters on the Internet requires a computer to have software capable of identifying the internal code of the text and to have the suitable fonts installed to look up the code to render and display the correct shapes [6].

It is difficult for Web developers to create and develop Web sites to suit all users in different countries. Each language usually encodes the scripts it uses in one set with code point assignment, which we call code character sets, or codeset for short. Some languages even have more than one coded character set. For example, there were three coded character sets for Chinese: Guobiao (GB) for Simplified Chinese, BIG5 and CNS for Traditional Chinese before Unicode was available. Usually, different fonts are designed for different codesets, such as the MingLiu font (細明體) that is designed for Big5, and the SimSun font that is designed for GB. In addition, Web pages in languages such as CJK are commonly used in Asian Web sites and they may be mixed together in the same page, which is referred to as a multilingual page [7]. For example, many Chinese

Web sites in Hong Kong contain multilingual pages with both Chinese characters and English words in the same page.

Language	Principal Locations	Population (in millions)
Chinese	China	885
English	North America, Great Britain, Australia, South Africa	450
Hindi-Urdu	India, Pakistan	333
Spanish	South America, Spain	266
Portuguese	Brazil, Portugal, Angola, Mozambique	175
Bengali	Bangladesh, India	162
Russian	Former Soviet Union	153
Arabic	North Africa, Middle East	150
Japanese	Japan	126
French	France, Canada, Belgium, Switzerland, Black Africa	122
German	Germany, Austria, Switzerland	118
Wu	China (Shanghai)	77
Javanese	Indonesia (Java)	75
Korean	Korea	72
Italian	Italy	63
Marathi	South India	65
Telugu	South India	55
Tamil	South India, Sri Lanka	48
Cantonese	China (Canton)	47
Ukrainian	Ukraine	46

Table 2-1: The world's 20 main languages

To increase the readership of the Web, Web developers usually create different language versions of the same Web content. In Hong Kong, many Web sites are prepared in two versions: English and Traditional Chinese. Sometimes they may even use Simplified Chinese because they mean to serve people both in Hong Kong and China. It is up to the user to decide which version to look at. A more desirable form of Web management is for the server to automatically obtain the font set installed on a client machine and to provide the user with the Web pages of the correct codeset encoding. That is, a machine having Traditional Chinese fonts will automatically obtain Web pages coded in Traditional Chinese, while machines containing simplified Chinese fonts will obtain the Web pages in Simplified Chinese.

Due to the rapid development of electronic devices in recent years, people can access the Internet any time and anywhere using a mobile phone or a Personal Digital Assistant (PDA), etc. These mobile devices are usually called thin-client devices. In this thesis, a thin-client refers to any computer or device that has limited resource for Web browsing. Such resource limitation can either be in hardware or software. We can thus further define hardware thin-client and software thin-client as follows:

Hardware thin-client – a device that has limited hardware resources such as limited memory or limited storage. For example, a mobile phone contains limited memory and it is impossible to install the whole set of CJK fonts.

Software thin-client – a device that has limited software resources. The device can be a small PDA but it can also be a Personal Computer (PC). The point is that it is incapable of handling CJK characters either due to the lack of CJK fonts and input method support or personal user preference to avoid possible software incompatibility. For example, for a computer installed with English Windows only, no CJK input and font display is supported although it has enough memory and storage,. Thus CJK characters cannot be displayed properly. There are also cases when people simply do not want to switch to a multilingual platform because they worry it may not be compatible with some software that they use for other work.

2.2 Current Methods for CJK Reading on Internet

There are several solutions to help users to read CJK character on the Internet. These solutions can be divided into three major categories: The first category is called *client-side solutions* that rely on setup and configuration on client machines such as the use of locally installed fonts [8, 9]. The second category is called *server-side solution*s that rely on character display provided by a server so that no configuration is needed on the client side. Examples include the Text-to-Image Replacement method on the server-side [10]. The third category is called *client-server solutions* that rely on the cooperation of both the client and the server to display CJK characters. An example is the Web Page Embedded Fonts method [11, 12].

2.2.1 Use of Locally Installed Fonts

Displaying CJK characters correctly requires a computer to have software capable of identifying the internal code of the Web page and to have suitable fonts installed so that it can look up the internal code to render and display the correct character images.

The basic method for reading CJK characters on the Internet is the use of locally installed fonts. Locally installed fonts can be provided in three ways: as *system fonts* provided by the local operating system, *browser specific fonts* provided by a Web browser, and *use of third-party CJK viewer.* Chinese Windows 98 is an example of an operating system that provides locally installed system fonts. Netscape Browser and the Internet Explorer can provide many specific fonts independent of the specific system they run on. NJ-Star is an example of a third-party CJK viewer that assistant users read the CJK text.

2.2.1.1 Locally Installed System Fonts

For operating systems localized for a specific language, such as the Traditional Chinese version of Microsoft Windows, Traditional Chinese characters (Big5 and extended Big5 encoding characters) are supported at the system level with appropriate fonts installed , such as MingLiu (細明體) and PMingLiu (新細明體). Machines running such an operating system have the ability to display CJK characters in all applications including Internet browsing [8]. When users access a Traditional Chinese Web site, the browser can display Traditional Chinese characters using the system fonts installed.

The advantage of this method is that all of the applications including the Internet browser can display CJK characters. However, if a user does not use a CJK system, he is not able to view any CJK characters because no pre-installed CJK fonts are available.

To display text in a different font face and type, users need to install the corresponding fonts. As a result, operating system vendors and font vendors must work together to localize the system and package the fonts for each localized system. Users can also purchase additional fonts for a localized system. If one Web site is designed to display in MingLiu (細明體) and another in Kaiu (標楷體), a user will have to install both the MingLiu and Kaiu to read the two different Web sites. Since Asian glyphs such as CJK characters are much more complex than those of an alphabet-based language, the CJK font file size is much larger than that of English. The default Chinese font (MingLiu) in Traditional Chinese Windows requires 8.23 MB whereas the English font (Times New Roman) requires only 315 KB. We can observe that the size of the Traditional Chinese font file is 20 times larger than the English fonts file. Therefore, memory requirements for the CJK font is much more demanding. Such large font files are not suitable for thin-client devices with limited resource since thin-clients most often rely on the network to provide computing resource and software [13]. With the development of Unicode, where 60,000 Chinese characters are supported, the font files will be much larger than Big5, for example, which only contains 13,000 Chinese characters.

In conclusion, locally installed system fonts require large memory space to store complete font files. Such a space requirement is often unrealistic in a hardware thin-client environment. For True Type font support under the Windows environment, the default font file (mingliu.ttc) for the Traditional Chinese environment requires 8.3MB, the default font file (simsun.ttc) for the Simplified Chinese environment is 10MB and the default font file (msmincho.ttc) for the Japanese environment is 8.7MB. If a user plans to use Unicode to replace the above three files, it would need a 20MB memory space for 20,902 characters. In addition, the memory needs to be much larger to support over 60,000 characters in Unicode 3.1 which is the latest version of the Unicode standard. Sometimes, the font file itself is already larger than the memory space provided in a thin-client device. For example, a standard Palm m100 only contains 2MB of memory, and it is almost impossible to install the CJK font files.

Besides, the procedure of font installation is not as easy as might be expected, especially in Macintosh, Unix and PDA systems. Many computer beginners do not know how to install fonts and how to configure the system even if they can obtain the fonts.

2.2.1.2 Locally Installed Browser Specific Fonts

For systems that are not specifically designed with CJK support, particularly in the English Windows environment, the use of browser specific fonts is a helpful solution. Most browsers support the display of certain language specific fonts independent of the system itself, provided that the corresponding fonts are made available inside the browser software.

This method normally requires users to download the related browser specific fonts and install them in the local machine. For example, if a user needs to read CJK Web pages, he/she must go to the site that supplies the browser's specific font file and download the required fonts and install them in his machine for the browser. After the system and the browser are properly configured, CJK characters can be displayed correctly in his browser. The most popular Internet browsers: Microsoft Internet Explorer (I.E.) makes use of this method to support CJK character display under all Windows platforms from version 4 onwards [9].

The advantage of this method is that the downloading and configuration only needs to be done once. However, it still requires users to choose the right font display each time they access a Web page using a different encoding. Also, the downloaded font file can only be used by a specific browser because different browsers may require different font files. Furthermore, this method provides limited font types. For example, there is only MingLiu for Traditional Chinese support in the Internet Explorer under the Windows environment. Besides, not all browsers support browser specific fonts. Currently, only Microsoft Internet Explorer supports browser specific fonts under the Windows environment.

2.2.1.3 Use of Third-party CJK Viewer

The use of third-party CJK viewer is another helpful solution for those systems that are not specifically designed with CJK support. By using these third-party CJK viewers such as NJ-Star and UnionWay, users can also read the CJK text in their non-CJK environment.

This method normally requires users to install a so-called "System plug-in" application and specific fonts in their local machine. For example, if a user wants to use NJ-Star to read CJK Web pages, he/she must install the viewer application along with the specific CJK fonts in his machine for the browser. After the system, font directory and the browser are properly configured, CJK characters can be displayed correctly in his browser.

The advantage of this method is the same of the locally installed browser specific fonts that the downloading, installation and configuration only needs to be done once. However, it still requires users to install the correct version of the third-party CJK viewer since these applications are Operation System dependent. Install and incorrect version of third-party viewer might due the to crush of the System. For example, "NJ-WIN CJK Viewer" does not fully support Windows XP. Moreover, most of these viewers only designed for Windows-based system, there are no any support on MAC, Unix or other operation system.

2.2.2 Text-to-Image Replacement

Text-to-Image replacement is a server-side solution in which HTML pages are converted to image(s) by using a proxy server before being sent to a client. There are two approaches for this replacement scheme. The first approach is called *Whole Page Replacement* in which the entire content within a Web page is converted into one large image and then sent to a client. The second approach is called *Character-to-Image Replacement* in which each CJK character is converted into one character image. In the latter case, a browser receives an HTML file instead of the image file used by the Whole Page Replacement scheme.

2.2.2.1 Whole Page Replacement

The Whole Page Replacement scheme converts the entire Web page to a single image before it is sent to a client [10]. When a user wants to access a CJK Web site but does not have a CJK platform, he/she first goes to a proxy site. The proxy that supports Whole Page Replacement converts the requested page into a large image and then sends it back to the user.

However, this scheme has some serious problems and it is not feasible for hardware thin-clients. For example, for a Web page with text occupying a 640x480 pixel space, which requires less than 1KB of memory for display on a screen, the resulting image file is as large as 50KB. Consequently, it takes quite a long time to generate the image file and then to download it. This delay is often very annoying for users [14]. In addition, complex structures in HTML documents such as the Control Button in the <FORM> tag cannot be handled. The converted image will disable the interaction related features for such a Web page preventing users from inputting data or giving feedback. Furthermore, searching of text in such an image is impossible. Therefore, the Whole Page Replacement scheme is not feasible for hardware thin-clients and often annoying to software thin-client users, although it may be helpful sometimes.

2.2.2.2 Character-to-Image Replacement

To overcome the limitation of the Whole Page Replacement Scheme, the so-called Character-to-Image Replacement Scheme was proposed. The basic idea of the Character-to-Image Replacement Scheme is the same as that of Whole Page Replacement except that the Character-to-Image Replacement converts only CJK characters into separate character images, which are embedded in an HTML page before being sent to the client.

Since duplicate characters in a page require only one such image to be embedded and embedded images in a Web page can be downloaded in parallel through the HTTP/1.1 protocol [15], the download time is much shorter than that of the Whole Page Replacement scheme.

2.2.3 Web Page Embedded Fonts

The latest Netscape Communicator and Internet Explorer can create dynamically downloadable font resources that encode the fonts the Web pages need so that they can be viewed as intended on machines without the fonts installed. The concept of Web Page Embedded Fonts is very simple. When Web developers or Web designers, who are called *Content Developers*, collectively create a Web page, a separate font definition file containing records of the characters and the font rendering information of only those characters used in the Web page is also prepared. When a browser downloads a Web page, the link to the font definition file is embedded in the HTML page. If the client does not have the right font, the client browser can automatically download the font definition file from the server and then render the character fonts on the client side. Currently, there are two Web Page Embedded Fonts specifications, one is the

Dynamic Font proposed by Bitstream and implemented by Netscape [11]; the other is the Web Embedding Fonts Tool (WEFT) developed by Microsoft and supported only on Internet Explorer version 4 and above [12].

2.2.3.1 Dynamic Font

The Dynamic Font approach is a Web Page Embedded Fonts method proposed by Bitstream. Instead of relying on the default fonts that users have in their browsers, content developers can now create pages using the fonts they have on their systems with the assurance that those pages will be displayed in a browser with the font formatting intact [16].

To create Web pages with Dynamic Fonts, a content developer needs to create a font definition file in PFR format, where PFR stands for Portable Font Resource and is defined by Bitsteam [17]. A PFR contains representations of all the characters in a particular font. For example, "Amelia.pfr" includes all the characters in the Amelia font. The content developer needs to use a <LINK> tag to link the HTML document with a "PFR" file and specify the fonts in the HTML document.

There are two ways to use Dynamic Fonts in an HTML document. The first way is to use the <FONT FACE> tag within an HTML document to specify the fonts. In the following example, content developers first specify the dynamic fonts file in the <LINK> tag, then make use of it through the <FONT> tag in the HTML document.

```
<LINK REL="FONTDEF" SRC="dynamicfont.pfr">
...
<FONT FACE="dynamicfont"> Text </FONT>
```

The other way is to use a Cascading Style Sheet (CSS) to specify the fonts where Cascading Style Sheets is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents. In the following example, the content developer first specifies the dynamic fonts file in the <LINK> tag, then uses the dynamic font by defining the style for the <P> tag.

```
<LINK REL="FONTDEF" SRC="dynamicfont.pfr">
...
<STYLE TYPE="text/css">
  P { font-family: "dynamicfont"; }
</STYLE>
...
<p> Text </p>
```

Although Dynamic Font can display CJK characters without relying on a client's locally installed fonts, a content developer must have the related fonts file and the WebFont Maker software (PFR file Maker) installed in his machine [17]. In addition, no browser can display Dynamic Font in a Web page without the installation of Bitstream's Font Displayer software (WebFont Player) such as the Character Shape Player [18] on the client, which may be a challenge to both types of thin clients.

2.2.3.2 Web Embedding Fonts Tool

The Web Embedding Fonts Tool (WEFT) is a stand-alone tool to help content developers to create embedded fonts that can be used when displaying their Web pages. WEFT allows content developers to create "font objects" that are linked to their Web pages so that when an Internet Explorer user views the pages he can see them displayed in the font style contained within the font object [19].

To use WEFT, a content developer needs to create a font object (.eot file) by using the "WEFT 3 wizard", and then he/she adds a STYLE section in the header section of each HTML page that uses one or more font objects. The code added should conform to the current Cascading Style Sheet 2 (CSS2) specification as shown in the following example:

```
<STYLE>
@font-face {font-family: "WEFTFont";
            src: url(WEFTFont.eot)}
H1 {font-family: "Verdana", "WEFTFont"}
</STYLE>
```

In the above HTML file, the code tells Microsoft Internet Explorer to use a user specific font object whenever the font "WEFTFont" is specified within the page. The browser will use the font object regardless of whether the font is specified using the <FONT FACE> tag or inline Cascading Style Sheet.

The major problem of WEFT is that this technique only supports Microsoft Internet Explorer with a built-in WEFT viewer. There is no uniform specification for Web page embedded fonts, and the current proposals only work in a browser that contains a built-in Web page embedded fonts rendering tool. The Web page embedded fonts approach also requires several iterations of client-server communication, which imposes a fixed communication delay that may not be acceptable for thin-client devices such as Wireless Application Protocol (WAP) phones.

2.3 Discussion on the three methods

The three methods mentioned in this chapter provide users with different ways to read CJK characters. Locally installed fonts and the Web page embedded fonts methods, however, are not suitable for the thin-client environment because of the excessive resource requirements of the former and the lack of standards compliance when implementing the latter.

For thin-client browsing, the most important issue is resource control. The font installation method can help users to read CJK characters on the Internet, but it requires a lot of space to store CJK fonts and is thus not feasible in the hardware thin-client environment. Although the Web page embedded font method requires only character fonts to be downloaded, no hardware thin-client browser supports this technology currently. There are systems which use Whole Page Replacement to help users to browse without the need for locally installed fonts. However, it is not suitable due to its large file size and long download time. Besides, it cannot handle complex structures such as control buttons in a form. Searching text in such an image is also impossible. It seems that the only viable method now for CJK character display in thin-client browsers is the Character-to-Image scheme. In this thesis we will focus on the character-to-image conversion technique only.

Based on the technology of the character-to-image scheme, some Character-to-Image Gateways have been developed for Web content browsing since 1995. Usually, a Character-to-Image Gateway is a Web-based mediator that converts the characters of the requested Web page to images before they are sent to clients. For simplicity, we use the term Gateway to refer to any character-to-image in the rest of this thesis.

The first gateway "HANZIX" was developed by Siu-Chi Hsu, Kin-Hong Lee, Qin Lu, Man-Fai Wong, and Wing-Shing Wong in 1995. It is an Unicode-based gateway that convert the input web page to Unicode format [20] and not yet employ the character-to-image technology.

In 1995, Ka-Ping Yee developed "Shodouka" which is a Web mediator that renders the Kana and Kanji of Japanese Web documents to character GIF images. It won the ACM Webbie First Prize because of its usefulness and popularity [21]. By using Shodouka, users can view Japanese documents without the need for locally installed Japanese fonts.

In 1996, Ka-Ping Yee also developed the "MINSE PolyMediator" for mathematical notation rendering [22]. MINSE PolyMediator aims to render whole mathematical and scientific expressions into a small image so that Web browsers need not rely on any special mathematical and scientific font.

In 1998, Silas Brown developed the Web Access Gateway based on the technology from Shodouka [23]. This is a very powerful gateway for the visually impaired and international users. Web Access Gateway now supports both Traditional Chinese and Japanese Web pages by converting them into images [24].

Although these gateways are helpful for Web content accessibility, there are several limitations in the current character-to-image gateway for the thin-client environment. Firstly, these systems can neither handle the <FORM> tag nor provide input methods for user input. Secondly, they require clients to know the encoding of the target Web page that they want to browse and manually choose the correct converter. Let us take Web Access Gateway as an example. If the target Web page is in Traditional Chinese, but the user does not know this, the gateway only takes the default Japanese characters to the image engine and thus the target page cannot be rendered successfully.

In Summary, a character-to-image gateway must be mindful of resource limitations as well as comprehensiveness to make it useful for practical purposes.

3. Design Objectives and System Framework

In this thesis, we aim to design a compressive gateway with full consideration for the limitations of a thin-client environment. The proposed gateway is called *Pelutech*, which stands for **PE**rspicacious and **LU**ciferous **TECH**nology. In this chapter, we first discuss the design objectives of Pelutech, and then present the system architecture and its components. We assume that Pelutech will handle character-to-image conversion only for HTML Web pages. At the end of the thesis, we present a brief discussion on how to remove this limitation and support other Web languages.

3.1 Design Objectives

3.1.1 Detection of Installed Fonts at the Client-side

In order to guarantee correct rendering of characters and graphical objects on a client machine, a gateway needs to know the type of fonts installed at the client side. Font detection is an unresolved issue in the current World Wide Web technology because there is no straightforward method to detect the types of fonts installed on the client machine due to security reasons [25].

Since no current system can successfully detect what fonts a client has installed on his/her machine, all current gateways provide full CJK character images to clients irrespective of their actual need. If a gateway knows all the client-supported fonts, the gateway can convert characters to images only based on need, eliminating unnecessary conversions. For instance, if the gateway knows the client has a Traditional Chinese font, then the gateway can pass the original page directly to the client rather than a converted (character-to-image) page. This would eliminate unnecessary conversion, and thereby reduce the workload placed on the gateway, as well as the download time and the user's waiting time.

Consequently, the first design objective of Pelutech is to have the ability to detect the type of fonts available on a client machine. With a successful detection algorithm, the gateway can convert characters into images only when necessary, thus saving resources and time.

3.1.2 Automatic Code-set Detection

The existing gateways require users to manually select the preferred Web page format, either text or graphics. If users think that they have proper fonts, they can choose the text format. Otherwise, they can choose the graphical format. The major problem with this approach is that it involves manual operation and it requires a user to know detailed information about the system and its configuration.

Methods must be provided in Pelutech to automatically detect character codesets and font sets supported in a client machine, so that manual selection can be eliminated. This is the second design objective of Pelutech.

3.1.3 Wide Style of Converted Image Support

Almost all images formed by the current gateways are static in that the display style and color are set by the gateway regardless of the original page specification; images cannot be produced in the users' preferred color and font face. For example, if the text in the original Web page were in blue Kaiu (仿宋

體), it would not be acceptable if the output character images were converted to black color MingLiu (細明體).

Most gateways only generate opaque character images without dealing with background color, occasionally resulting in unclear and messy Web pages. Figure 3-1 shows an original Web page and Figure 3-2 shows the corresponding converted Web page with opaque character images.

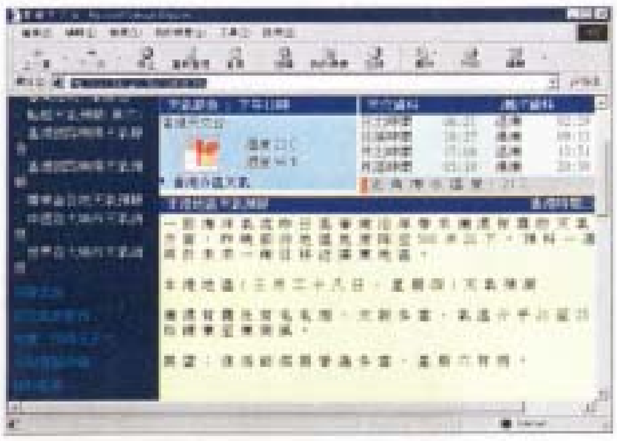


Figure 3-1: Original page for the Hong Kong Observatory

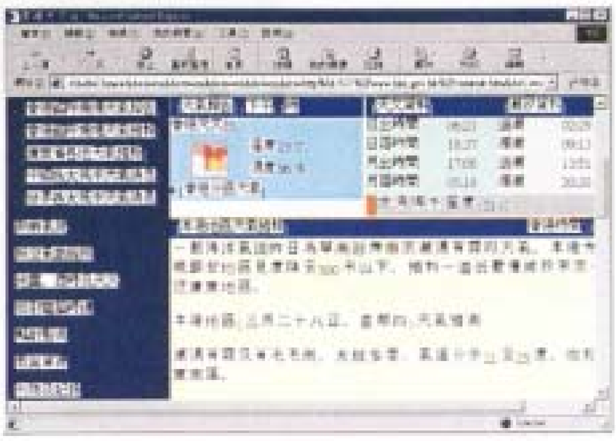


Figure 3-2: Untidy page after conversion

It is our design objective to generate transparent character images as well as character images for various kinds of font face, type, color, size, and style. The formatting and the appearance of reformatted Web pages should look similar to the original. Otherwise, the conversion may lose part of the original information because font style and color often provide visual cues for users to interpret the document [26].

3.1.4 Handling Complex Structure in an HTML Document

Nowadays, Web sites have become much more complex. Many Web sites use not only HTML in Web documents, but also JavaScript, Dynamic HTML and Cascading Style Sheets (CSS) to enhance the interaction between the client and the Web site.

The first difficulty the current gateways face is how to handle the poorly formatted structure of the HTML language. For example when using the table in a form, the HTML file can be written in well-formatted syntax as shown here:

```
<form>
  <table>
  ...
  </table>
</form>
```

However, some Web pages may not be so well formatted, and we could find a switch in the closing order of the <form> and <table> tags as shown below:

```
<form>
  <table>
  ...
  </form>
</table>
```

Moreover, an HTML document can also be written without closing the <form> and <table> tags:

```
<form>
  <table>
  ...
```

Poorly formatted HTML documents increase the difficulty of parsing these documents before character-to-image conversion can be carried out. Figure 3-3 shows the original page for HongKong.com and Figure 3-4 shows the result when the Web Access Gateway fails to parse this Web site.

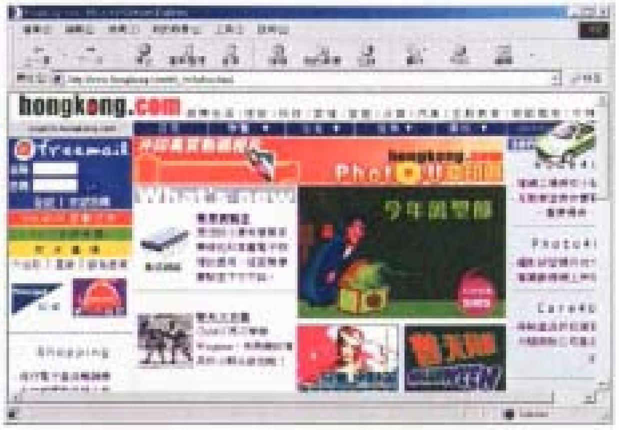


Figure 3-3: Original page for the HongKong.com

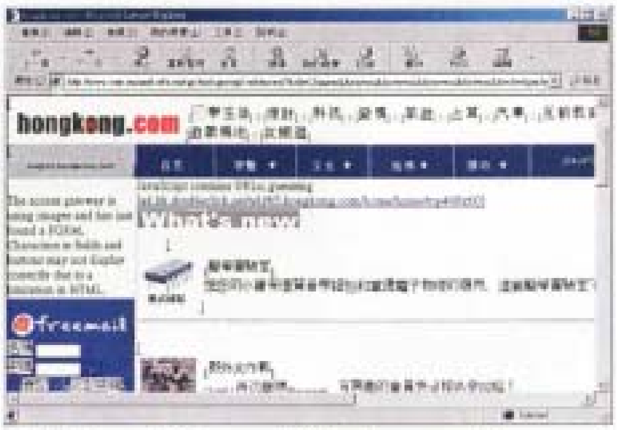


Figure 3-4: Web Access Gateway failing to parse the complex structure

In addition, current gateways have only limited methods to parse HTML tags, and almost all are incapable of handling form controls provided by the <FORM> clause. The <FORM> tags such as Text Box, Submit Button and Text Area in an HTML document are often used to provide user interaction and selection during browsing. These <FORM> controls normally have text items that are displayed using local fonts. Currently, no browsers can display image items inside a control. The text-to-image conversion technique, however, is based on the replacement of each character with the corresponding character image. Converting the input selection items from the <FORM> control would devastate the display of these buttons on the client-side, since the buttons cannot take images as their input on the client machine. Therefore, all existing gateways skip the conversion of the <FORM> controls. However, a comprehensive system must have the capability to process CJK character items, otherwise, it would not be very useful.

It is our fourth design objective to handle the HTML form controls. If the label of a button control or a list box contains CJK characters, these form controls should be specially treated during page transformation.

3.1.5 Enable Text Searching in a Converted Document

In common Internet browsers, a user can search text within a Web page or a frame that the user is currently reading. The search algorithms used search the source code of the HTML page, and a browser built-in search program also highlights the matched words in the browser display. However, when characters are converted to images, a browser's built-in search utility no longer works because it is impossible to search character images in current implementations [27]. The major challenge for this type of searching is that the original Web content is converted to images. This means that the word "CAT" might be converted to the following format:

```
<IMG SRC = "C.GIF"><IMG SRC = "A.GIF"><IMG SRC = "T.GIF">
```

There is a need to investigate some methods to support searching that can handle converted images. This is our fifth design objective.

3.1.6 Provide Native Input Method Engine Support

Native Input methods are software components that map an input sequence into an internal code representation of the text [28]. The most convenient and common way of inputting text is typing on a keyboard. Although writing characters by mouse or digital pens is also popular. There are over ten thousand CJK characters and each character input requires a keyboard input sequence to map to its internal code.

Operating systems designed for the CJK environment are often equipped with the Input Method Engine (IME) for CJK character input; for example, Traditional Chinese Microsoft Windows has an IME. However, non-CJK platforms often do not have such support.

With the growth of the Internet and the increased mobility of information users, people can no longer be assumed to have access to systems that have their native language support at all times. Besides, Web browsing is not just a simple one-way communication where information is obtained from the server. A client and a server need to communicate with each other, and Web pages are a means for the server to obtain information from the client machines too. For instance, a user may submit a search request to a search engine, post messages and chat on the Internet using CJK characters. As a result, the provision of an input method becomes quite vital. However, there is no native input method support in the existing gateways.

For those systems which are not equipped with an input method engine, users can only install a third party software application or download a vendor's language pack for the operating system. However, the configuration of the third party software requires disk storage space and also good computer knowledge from the users. In addition, due to the popularity of Microsoft Windows, almost all third party software developers only design input method modules for Windows. In other words, other operating systems or platforms, like UNIX and Macintosh, are often neglected.

There is a need to provide a server-side input method engine that allows user text input through the Internet. The provision of this IME must allow text input in all form controls.

3.2 Pelutech Gateway

3.2.1 System Overview

Pelutech acts as a proxy server between a Web server and a client. When a client requests a Web page to be directed to Pelutech, Pelutech sends the request to the server and obtains the returned Web pages. After the original Web page is processed and transformed, Pelutech sends the reformatted Web page to the client. The flow of control is indicated in Figure 3-5 in next page.

In addition, Pelutech and the Web server are not necessarily to host on different machines. Pelutech can also be installed at a regular Web server to provide value-added services. But most likely, the proxy will reside on a third party host machine to provide independent service.

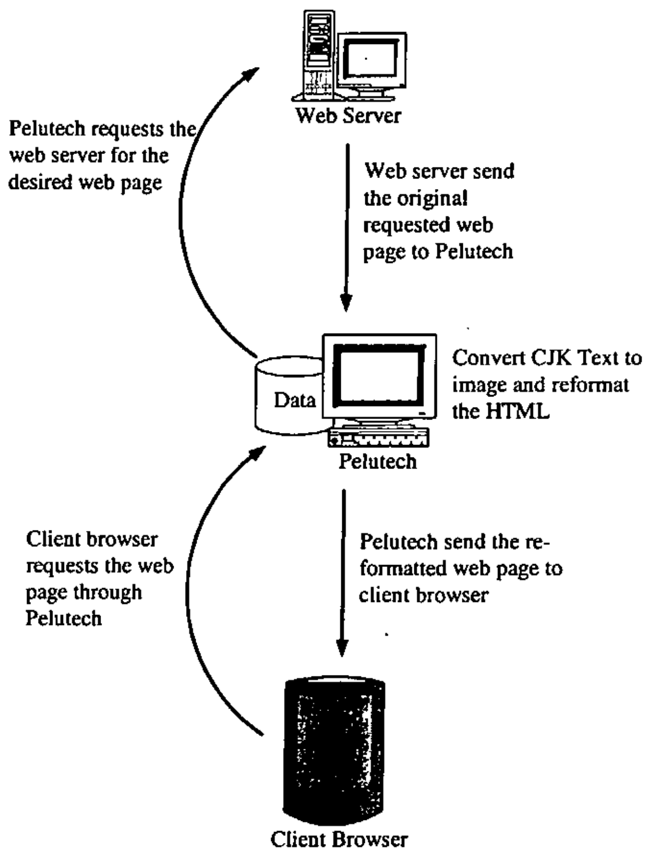

Web Server
Pelutech requests the web server for the desired web page
Web server send the original requested web page to Pelutech
Data
Convert CJK Text to image and reformat the HTML
Pelutech
Client browser requests the web page through Pelutech
Pelutech send the re-formatted web page to client browser
Client Browser

Figure 3-5: Flow of control among Pelutech, Client and Web Server

3.2.2 System Framework

Based on the functionalities to be provided by Pelutech, the system is designed to have three subsystems: the *Conversion Engine*, the *Input Method Engine* and the *Search Engine*. The Conversion Engine is a character-to-image conversion server that converts a Web page with CJK characters into a Web page with embedded CJK character images. The Input Method Engine is an interface that allows users to input CJK characters without any other input method installed on the client side. The Search Engine is used to provide character search functionality in a page with embedded character images.

Furthermore, the Conversion Engine has six components: the Locale Detector, the HTML Document Parser, the Codeset Detector, the Character Image Constructor, the Output Document Composer, and the Form Control

Builders. The Locale Detector is used to detect the font set supported on a client machine. The HTML Document Parser analyzes the incoming HTML documents. The Codeset Detector detects the encoding in the requested Web document. The Character Image Constructor produces the CJK character images. The Output Document Composer combines all components and forms a new HTML page. The Form Controls Builder constructs the <FORM> controls that interact with users. Obviously, it is used only if there are <FORM> clauses in the document. The functionalities of each component are summarized in Table 3-1 below for easy reference.

Component	Functions
Locale Detector	Detect the font support on the client machine.
HTML Document Parser	Parse the original HTML source file.
Codeset Converter	Detect the correct codeset of the requesting document.
Character Image Constructor	Construct the CJK character images.
Output Document Compositor	Reformat the Web page with the embedded character images.
Form Controls Builder	Convert the original HTML form control to a format that can be used by the client.

Table 3-1: Summary of the functions of the Conversion Engine Components

The Input Method Engine is responsible for the interaction between the user and Pelutech. It allows the user to input CJK characters when filling in forms and when searching, without installing any third party input method. It is delivered to the client machine whenever the client browser makes a request for its services.

The search engine provides not only the text content search function but also the image search function for the reformatted Web page. In this project, the client and the server approaches are studied to implement the search engine. This component can either reside at the client machine with the Input Method Engine or at the Pelutech server.

The overall System Framework of Pelutech is shown in Figure 3-6. Note that, in the figure the search engine first requires input. Thus the Input Method Engine interacts with the Search Engine directly. In addition, since form filling requires input, the Input Method Engine also interacts with the Form Controls Builder. Therefore, the search engine and input method interface are embedded into the converted Web page for the client to access.

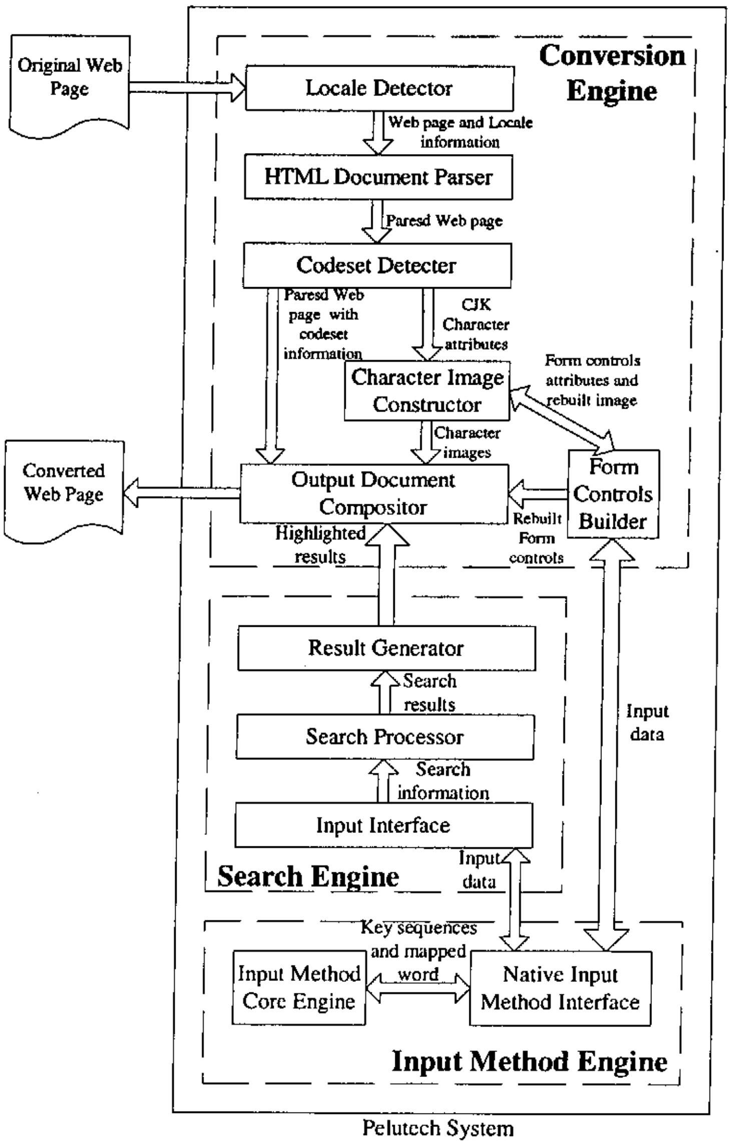


Figure 3-6: System Framework of Pelutech

4. Locale and Codeset Detection

In order to display Web content correctly, we must first discover ways to identify not only the encoding of a Web page but also the encoding and fonts available on the client side. Such information can then be exchanged and obtained during server-client communication [29]. There are two prerequisites for displaying a specified font correctly using the locally installed fonts. Firstly, the font must be installed on the client machine. Secondly, the browser must be able to locate the font and get the Web page displayed. The term *Locale* collectively refers to an encoding and its supported font sets for a given language environment. Once this locale information is available, the gateway can proceed with the conversion. The locale information, if available, can also help the gateway to avoid unnecessary conversions.

4.1 Related Locale Identification Technique

HTML documents for different languages are coded using different codesets. Even the same language may be coded in different codesets. For instance, Traditional Chinese can either be encoded in Big5, CNS, or Unicode [30]. In order to perform character-to-image conversion, two pieces of information must be obtained before proceeding to conversion. Firstly, we must know in what codeset an HTML document is encoded. Secondly, we need to detect the locale supported on the client machine. Early versions of the HTML specification do not contain any mechanism to detect the codeset of a Web page and all pages are assumed to be in ISO-8859-1 [31]. Starting from the HTML/3.0 specification, <Meta> tag allows content developers to announce the codeset of a text document. For example, a BIG5 encoded document can be announced as:

```
<meta http-equiv = "Content-Type" content = "text/html;
charset = big5">
```

Based on this codeset announcement, an Internet browser can find out in which codeset the Web document is encoded. Consequently, the client machine can display the contents correctly if it contains the required font. This method, however, requires the codeset announcement to be made explicitly in every Web page, which is not often found in existing Web pages.

The encoding information can be exchanged through the HTTP protocol based on two assumptions: (1) a client machine knows what codesets are provided locally, and (2) a Web server is aware of the encoding of the Web pages it maintains, either by default or through other means including codeset announcement in HTML. From HTTP version 1.1, a Web client and a Web server can place the encoding information in the header of the protocol [32]. There are two header fields for this purpose: ACCEPT-CHARSET and ACCEPT-LANGUAGE. The former specifies more precisely the codeset information whereas the latter is specific to the language (but not the codeset) and provides more flexibility. The syntax is given below:

```
Accept-Coded character set = "Accept-Coded character set"
":" 1#( ( coded character set | "*" ) [ ";" "q" "=" qvalue ] )

Accept-Language = "Accept-Language" ":" 1#(language-range
[";" "q" "=" qvalue])

language-range = ((1*8ALPHA*("-" 1*8ALPHA)) | "*")
```

Please note that the 3rd item "language-range" specifies a language range that is particularly useful for European Latin-based languages.

With the help of these two fields, a Web server knows the supported codesets in the client browser side. Based on such information, it can find out which character sets users can accept and convert the text to images only when necessary [33].

In the current implementation of the HTTP protocols, information can be exchanged only if users have the right browsers and have configured the browsers correctly. Even in a Chinese system such as Chinese Windows, browsers such as Netscape 4.x will not accept Chinese if users do not configure Netscape manually in the right fashion. In addition, the codeset setting in HTTP is done manually by users and they are not necessarily linked to any font sets. Thus it is possible that a browser indicates that it can accept GB, but GB fonts are not available locally. Therefore, the most direct way to obtain supported locale information on the client side is to detect the installed font from the client machine. Once the Web server gets the information, it can provide the page with the right encoding.

4.2 Locale Detection

Current World Wide Web technology does not provide any mechanism to do automatic font set detection. As servers do not have the right to access system resources such as data in the font directory and font registry, there is no direct method for Web servers to identify installed fonts on client machines. In this thesis, we have devised a novel method to do locale detection by probing client machines to get locale information automatically through a mechanism built into HTML.

4.2.1 Design Principles

Our method is based on the characteristics of the True Type font. Generally, a True Type font can be measured by its width, which is often called a font metric [34]. We assume that the gateway is aware of the names of different True Type font faces, such as MingLiu and SimSong, which sometimes are simply referred to as *fonts*. We also assume that different fonts for the same character have different widths. Thus, the same text would have different widths when displayed by different fonts.

In the Web browsing process, HTML allows font display information to be specified in Web pages. If the specified font in a Web page is not available, browsers are allowed to find a substitute font for display [35]. Both Internet browsers, Netscape and Internet Explorer, support the (so-called) *browser-instigated substitution* of a font if a Web page specified font does not exist on the client machine. The substitution proceeds as follows: To display a Web page in a specified font like MingLiu, the browser must first check whether the font exists on the client machine. If the specified font is found, it will be used to display the Web page. Otherwise, the browser will use the browser's default font such as Arial to substitute for the specified font MingLiu. We can also opt for an explicit substitution provided by HTML as follows:

```
<SPAN style = "font-family: MingLiu, Arial">FontDetect
</SPAN>
```

where the text "FontDetect" is to be displayed in MingLiu, and if it is not available it will be substituted by Arial [34].

In order to find out the font information, we make use of the browser-instigated substitution mechanism and the explicit substitution provided by HTML. We use the detection of font width to draw a conclusion on whether a client has installed the right font. To make the method feasible, we create two display elements to be inserted into an HTML page. The first element has a predefined text "FontDetect" with a predefined font Arial as shown below:

```
Element One:
  <SPAN style = "font-family: Arial">FontDetect</SPAN>
```

The second element contains the same text, but the specified font (or the font family) should be the one we want to detect as shown below:

```
Element Two:
  <SPAN style = "font-family: MingLiu, Arial">FontDetect
  </SPAN>
```

The first item is the name of the font we are checking MingLiu. The second is the name of the substitute font Arial, which should be the same font that was specified in the first element.

Since an Internet browser will use the substitute font if the specified font does not exist on the client machine, MingLiu will be substituted by Arial if it does not exist. As a result, the width of the second element will then be equal to that of the first element. However, if MingLiu does exist, the width of the second element will be different.

4.2.2 Design Details and Implementation

To carry out this method, we must, however, implement the font probing according to different style sheets supported by different browsers [36]. Thus the probing must also first identify the browser type and version.

Two environment variables are used to help us getting the browser type, one of them is *navigator.appName* and the other is *navigator.appVersion*. The function for getting the values of these variables is given in Figure 4-1, where a return value of 1 (Line 9) indicates Microsoft Internet Explorer 4 or above; a value of 2 (Line 7) indicates Netscape Communicator 4.x and a value of 3 (Line 5) indicates Netscape 6.x or Mozilla 0.x.

```
1. function CheckBrowserVersion() {
2.  var BrowserType = navigator.appName;
3.  var BrowserVersion = parseInt(navigator.appVersion);
4.  if (BrowserType=="Netscape" && BrowserVersion>=5)
5.   return 3;                  // Netscape 6.x or Mozilla
6.  else if (BrowserType=="Netscape" && BrowserVersion=4)
7.   return 2;                  // Netscape 4.x
8.  else if (BrowserType=="Microsoft Internet Explorer"&&
    BrowserVersion>=4)
9.   return 1;                  // IE 4.x or above
10. else
11.  return 0;                  // Others browsers
12. }
```

Figure 4-1: The Program Code of the Client Browser Analysis

When probing for font types, we use the <LAYER> mechanism provided in HTML to compare the font metrics, as shown in the function listed in Figure 4-2. Since Netscape 6.x and Internet Explorer do not support the <LAYER> Tag, <DIV> and <SPAN> are used to carry out the probing. Lines 2 to 9 show the code for the font detection used in Microsoft Internet Explorer; Lines 10 to 18 show the code for Netscape Communicator 4.x and Lines 19 to 26 show the code used in Netscape 6.x. Line 28 is used to compare the difference in width between two layers and return the result.

```
function CheckInstalledFont(FontName, BrowserVersion) {
 if (BrowserVersion==1) {    // IE 4.x or above
  if (!window.HTMLObject0) {
    document.write("<DIV ID=HTMLObject0 STYLE='position:absolute;
    width:auto; visibility:hidden; font:12pt Courier'>
    FontDetect</DIV>");
    document.write("<DIV ID=HTMLObject1 STYLE='position:absolute;
    width:auto; visibility:hidden; font-size:12pt'>
    FontDetect</DIV>");
  }
  HTMLObject1.style.fontFamily = FontName +", Courier";
  width0=HTMLObject0.offsetWidth;
  width1=HTMLObject1.offsetWidth;
 } else if (BrowserVersion==2) {// Netscape 4.x
  if (!window.HTMLObject0) {
    document.write("<LAYER VISIBILITY=HIDE><FONT FACE='Courier'
    POINT-SIZE=12> FontDetect</FONT></LAYER>");
    HTMLObject0 = document.layers[document.layers.length-1];
    document.write("<LAYER VISIBILITY=HIDE><FONT FACE='"+
    FontName +",Courier' POINT-SIZE =12>FontDetect</FONT>
    </LAYER>");
    HTMLObject1 = document.layers[document.layers.length-1];
  }
  width0=HTMLObject0.clip.width;
  width1=HTMLObject1.clip.width;
 } else if (BrowserVersion==3) {// Netscape 6.x
  if (!window.HTMLObject0) {
    document.write("<DIV ID=HTMLObject0 STYLE='position:absolute;
    width:auto; visibility:hidden; font:12pt Courier'>
    FontDetect</DIV>");
    document.write("<DIV ID=HTMLObject1 STYLE='position:absolute;
    width:auto; visibility:hidden; font-size:12pt'>
    FontDetect</DIV>");
  }
  document.getElementById("HTMLObject1").style.fontFamily=
  FontName+",Courier";
  width0 =
  parseInt(document.getElementById("HTMLObject0").offsetWidth);
```

```
    width1 =
    parseInt(document.getElementById("HTMLObject1").offsetWidth);
}
return (width0!=width1);    // compare width different and return
}
```

Figure 4-2: The Core Program for the Font Detection

To detect different font types, the probing must be done for each True type font. Thus the program must repeat the question "Have you installed MingLiu?" for each font name. Also, we must probe for the encoding of the font. In other words, we need to know the locales supported. Thus the question should really be "Have you installed Traditional Chinese font: MingLiu?" For this reason, the server must keep a table containing the names of all the basic True Type fonts required to display a specified coded character set. For example, to display Traditional Chinese correctly, the table must have entries for MingLiu or Kai. Information on SimSun, Microsoft Sans Serif or SimHei must also be available for viewing Simplified Chinese. A sample table is shown in Table 4-1 and a sample program for the installed font detection is shown in Figure 4-3. This sample program tests the font in the array-based database "TestFont" sequentially and returns TRUE when the font is found on the client machine.

Coded Character Set	**Required True Type Fonts**
Traditional Chinese	Ming Liu
	Kaiu
Simplified Chinese	SimSun
	Microsoft Sans Serif
	SimHei
Japanese	MS Mincho
Korean	Batang
	Gungsuh

Table 4-1: Part of the Codeset-required Font Table

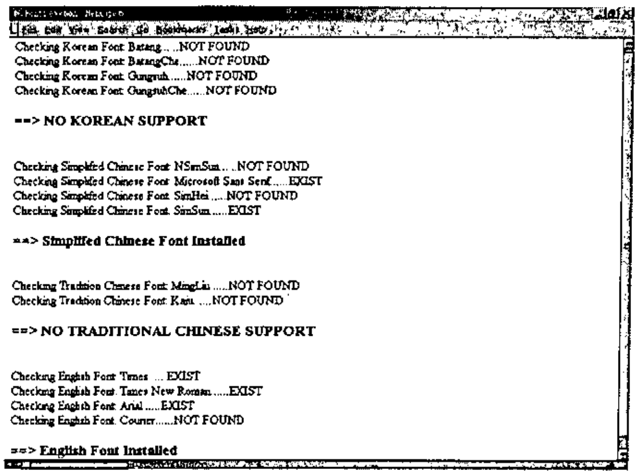
```
var BrowserVersion=BrowserChecker();
var TestFont=new Array("Batang", "Gungsuh");

for (i=0;i<TestFont.length;i++) {
   if (checkIfInstalled(TestFont[i], BrowserVersion)) {
      return True;
   }
}
```

Figure 4-3: The Program Code of the Installed Korean Fonts Detection

4.2.3 Experimental Results and Discussion

To evaluate the accuracy of this locale detection algorithm, we have performed the following experiments. We set-up a Web site that contained the Locale Detector, and invited 100 people, who were from Hong Kong, China, Taiwan, Japan, Korea, Malaysia, United State and Canada, to visit this Web site and give feedback on the tested results.

In this experiment, the evaluators were asked to enable the JavaScript option in their browsers before they visited our test Web site. Then they sent back the result and reported their system configuration to us. We found that the accuracy was as high as 99%. The sample screen and the results are shown below:



```
Checking Korean Font: Batang.. ..NOT FOUND
Checking Korean Font: BatangChe......NOT FOUND
Checking Korean Font: Gungsuh.....NOT FOUND
Checking Korean Font: GungsuhChe......NOT FOUND

==> NO KOREAN SUPPORT

Checking Simplifed Chinese Font: NSimSun.. ..NOT FOUND
Checking Simplifed Chinese Font: Microsoft Sans Serif......EXIST
Checking Simplifed Chinese Font: SimHei .....NOT FOUND
Checking Simplifed Chinese Font: SimSun......EXIST

==> Simplifed Chinese Font Installed

Checking Tradition Chinese Font: MingLiu .....NOT FOUND
Checking Tradition Chinese Font: Kaiu ....NOT FOUND

==> NO TRADITIONAL CHINESE SUPPORT

Checking English Font: Times ... EXIST
Checking English Font: Times New Roman......EXIST
Checking English Font: Arial......EXIST
Checking English Font: Courier......NOT FOUND

==> English Font Installed
```

Figure 4-4: Screen shot for a tested page

Operation System	Number of People	Font Detection Result	
		Success	Fail
Windows CE	1	1	-
Windows 3.1	1	1	-
Windows 95	12	12	-
Windows 98	35	35	-
Windows ME	29	29	-
Windows 2000	20	20	-
Mac OS 8	1	1	-
Linux	1	-	1

Table 4-2: Experimental Results via Operating Systems

Internet Browser	Number of People	Font Detection Result	
		Success	Fail
Netscape 4.x	29	28	1
Netscape 6.x	3	3	-
Internet Explorer 4.x	10	10	-
Internet Explorer 5.x	46	46	-
Internet Explorer 6.x	1	1	-
Pocket IE	1	1	-

Table 4-3: Experimental Results via Internet Browsers

From Table 4-3, we can see that the locale detection results are very good. There is only 1% failure in each experiment. The failure occurred only in a Linux system because it was not using True Type fonts. For the failure in the Netscape 4.x, it was due to the evaluator not having followed our instructions and that JavaScript was switched off in the browser.

4.3 Encoding Detection of Web Pages

For matching the encoding of a Web page with the locale on a client machine, we also need to detect the encoding of the Web pages to be displayed. As mentioned above, the <META> tag allows content developers to announce the codeset of the text data. Therefore, it is not difficult to identify in which codeset it is encoded if a Web page contains the character set definition in the Meta tag as shown below:

```
<META HTTP-EQUIV = "Content-Type" CONTENT = "text/html;
charset = big5">
```

However, when Web pages do not contain the character set definition (the CHARSET attribute in the <META> tag) we have to use heuristics to discover the character set.

4.3.1 Design of the Codeset Detector

This Codeset Detector contains two codeset analysis components. The first one is used to retrieve the CHARSET announcement in the HTML document header [37]. The second one is used to analyze the codeset based on the domain of the requested Web page. The system flowchart is shown in Figure 4-5.

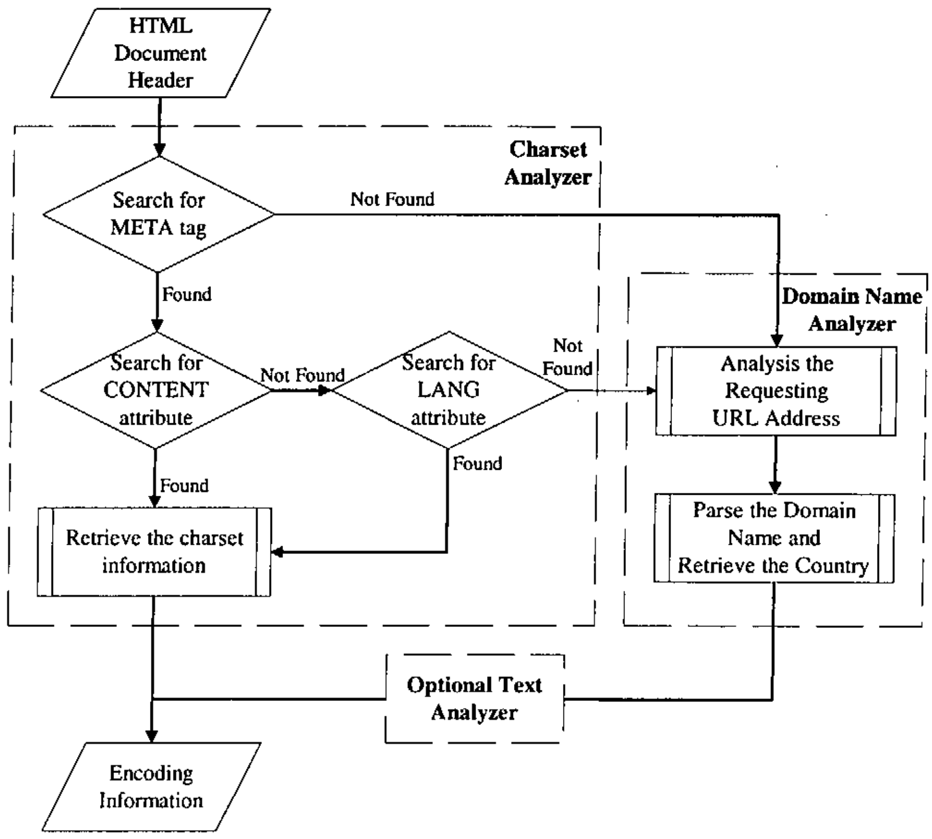

HTML Document Header
Charset Analyzer
Search for META tag
Not Found
Found
Search for CONTENT attribute
Not Found
Search for LANG attribute
Not Found
Found
Found
Retrieve the charset information
Domain Name Analyzer
Analysis the Requesting URL Address
Parse the Domain Name and Retrieve the Country
Optional Text Analyzer
Encoding Information

Figure 4-5: Flow chart of the Codeset Detector

After the HTML document that is based on a user's request is obtained, the HTML document header will be retrieved and parsed. If the Codeset Detector can successfully find the <META> tag, it will search for "CONTENT" attributes and identify the **CHARSET** value. It will also try to search for the "LANG" attributes in case it fails to find the **CHARSET** value.

If there is neither a <META> tag nor a codeset announcement in the HTML document header, the requested URL address will be analyzed. The detection will take both the URL and the IP address into consideration. The Codeset Detector will retrieve the local domain name (e.g. http://www.info.gov.hk) from the complete URL address (e.g. http://www.info.gov.hk/cisd/cwhatsne.htm). Then it will detect the country specified keyword (.hk) from the domain name and then check the encoding against a country database provided at the gateway.

Since some countries might have more than one encoding, for example, Big5, GB and Unicode are commonly used together in the website of Hong Kong, an optional text analyzer can install to enhance the accuracy of the encoding detection.

Domain Name	**Country**	**Possible Encoding**
.jp	Japan	Shift_JIS
.kr	Korea	EUC-KR
.tw	Taiwan	BIG5
.hk	Hong Kong	BIG5
.cn	China	GB
.ca	Canada	ISO-8859-1

Table 4-4: Part of the database for domain name via possible encoding

4.3.2 Experimental Results and Discussion

To evaluate the different possibilities and the accuracy of the detection algorithm, we conducted an evaluation test. We randomly picked 100 Web sites that are in Traditional Chinese, Simplified Chinese, Japanese, Korean and English, 20 for each language. Then we used the Codeset Detector to predict the encoding for the requested Web pages. The result is shown in Table 4-5.

Web Site Encoding	**Number of Web sites**	**Testing Result**	
		Success	**Fail**
Traditional Chinese	20	18	2
Simplified Chinese	20	19	1
Japanese	20	17	3
Korean	20	20	-
English	20	20	-

Table 4-5: Experimental Results for Codeset Detection

We found that the accuracy of the result is as high as 95%. Although the result is not 100% accurate, it is useful as an additional heuristics in codeset detection and the algorithm is very simple for practical use. After evaluation, we found that the failed Web sites were the personal Web sites hosted by free Web servers such as Geocities.com without any codeset announcement in the HTML document. Then our system analyzed the domain name and returned "Traditional Chinese" due to the end string ".hk" in the domain. However, when we manually browsed the Web sites, we found that it was a Simplified Chinese Web site and we found that even the Web browser could not correctly detect the codeset and it returned unreadable characters on the screen. To resolve this problem, a more comprehensive and automatic codeset algorithm based on Web content can be easily added. However, the processing time will be greatly increased and because of that it is not worth implementing.

Locale detection and Web page codeset detection require less than 1/10th of a second to probe a client machine. Even with this delay, it is still a worthwhile function compared to blindly doing character-to-image conversion. The major reason is that by using Locale Detection and Codeset Detection, we can retrieve locale information from client machines. Thus, whenever the required locale is available in the client machines, we can give users the original Web documents rather than the converted character images. This can greatly reduce server-processing time and minimize server workload.

5. Construction of Character Images

5.1 System Flow Control

The *Character Image Constructor*, or Constructor for short, is designed to generate a variety of character images. Once the Character Image Constructor obtains the font face required, it will locate the appropriate True Type font files and extract the character features. The constructor then uses the internal code of the character and other attributes such as font size, font color, background color and font type to render the character image.

The generation of a character image is divided into five steps. Firstly, the Constructor obtains the internal code and attributes such as style, color and size of the character. Secondly, the Constructor will convert the internal code of the character to Unicode by using a Unicode Mapping Table. The reason for Unicode conversion is that it supports nearly all characters. Since Unicode can support all Eastern and Western characters the system can be extended easily for other languages besides CJK characters. Thirdly, the system will generate a skeleton image based on the specified size. For example, if a character is of the size 12 points, a 16 x 16 pixels raw bitmap will be created. Fourthly, the Constructor extracts the character features from the True Type font file, and finally it renders the character image by using a True Type font to bitmap image conversion function.

5.2 Character Image Naming Specification

In order to maintain information on a character such as its style, size and color after it is converted into an image, we must incorporate such information into the character image [38]. To facilitate searching, we also need to include the internal code in the image. Unicode is deliberately chosen as the internal code because our Pelutech Gateway is designed to support CJK characters. For the Chinese character "中", which is U+4E00 in Unicode, Pelutech will name it as "4E00.gif". In fact, to incorporate font, background color and font size into the rendered character image, all the information required will be embedded into the constructor script. The format is shown below:

```
f=[Internal      Code]&font=[Font      Face]&style=[Font
Style]&size=[Font  Size]&fc=[Font  Color]&bf=[Background
Color]
```

In the above specification, *f* is used to specify the internal code (Unicode) of the character; *font* is used to specify the name of the font face; *style* is used to specify the font style such as italic, or bold; *size* is used to specify the font size in points; *fc* and *bc* are used to specify the font color and background color in Hex code, respectively. For example, the Chinese character "中" in black font color (#000000), using the MingLiu font face, with Underline style and the default font size (12 points) with white background color (#FFFFFF) page is scripted as "f=4E00&font=MingLiu&size=12&style=U&fc=000000&bc=FFFFFF" in the Character Image Constructor.

5.3 True Type Font File Specification

In order to convert a character to an image, we must first understand the True Type font file format.

The TrueType font file consists of a sequence of concatenated tables. Each table must be aligned and padded with zeroes if necessary [39]. The first table is called the *font directory*, a special table that facilitates access to the other tables in the font file. The directory is followed by a sequence of tables containing the font data. These tables can appear in any order. Certain tables are required for all fonts. Others are optional depending upon the functionality expected of a particular font. The tables have names known as tags. Currently defined tag names consist of four characters. Tag names with less than four characters have trailing spaces. When tag names are shown in text, they are enclosed in straight quotes. Tables that are required must appear in any valid TrueType font file [40]. The required tables and their tag names are shown in Table 5-1.

Tag Name	**Table name**
cmap	Character to glyph mapping
glyf	Glyph data
head	Font header
hhea	Horizontal header
hmtx	Horizontal metrics
loca	Index to location
maxp	Maximum profile
name	Naming
post	PostScript

Table 5-1: Table Name via Tag Name

After the Constructor obtains all required information for a character in a True Type font file, the Constructor can convert the True Type font, which is a vector-based outline font, to a bitmap-based image. Figure 5-2 shows an example of a converted bitmap image of the character "一" from the True Type font file MingLiu.

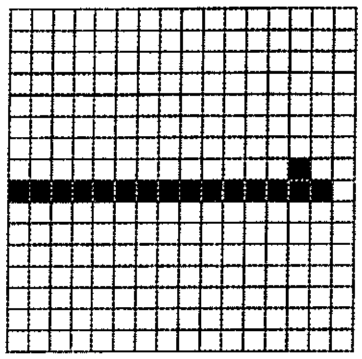


Figure 5-1: Chinese character ONE 16 by 16 bitmap

5.4 True Type to Bitmap Conversion

There are three steps to convert the True Type font information to browser-supported Web image: (1) retrieve information from True Type font file; (2) render a draft bitmap, then convert the bitmap to a browser-supported transparent image such as a GIF; and (3) compress the image for Internet transfer.

Firstly, we use the *Free Type Library* [41] to retrieve information from the True Type font file. The Free Type Library is a software font engine that is designed to be small, efficient, highly customizable and portable, while also being capable of producing high-quality output. It is a font service such as text layout or graphics processing. It greatly simplifies these tasks by providing a simple, easy-to-use and uniform interface to access the content of font files.

Secondly, we use the *GD Graphic Library* [42] for the True Type to bitmap conversion. The GD Graphic Library is a graphics library that is helpful for coding quickly and also for drawing images that are complete with lines, arcs, text, multiple colors, graphics cut-and-paste from other images, and flood fills that are particularly useful in World Wide Web applications.

With the help of the GD Graphic Library, the rendering of the character image can be simplified because the function *gdImageChar* can be used to automatically generate the character images for different sizes and font types. However, this function does not support CJK characters, which are 2 or more bytes. To overcome this problem, the function *gdImageStringFT* is used to input each CJK character as a UTF-8 string.

Since we need to reduce the image size for Internet transfer, we use the official PNG reference library - *libpng* [43], and the general-purpose compression library – *zlib* [44] for handling image compression because zlib is a loss-less data-compression library for use on almost every computer platform and operating system.

5.5 Sample Output

In order to show the comprehensiveness of our Constructor module, we show a list of examples of different character images that it generated. Tables 5-2 to 5-5 show the respective character images generated by our Constructor for the English alphabet, for numbers and symbols, for Traditional and Simplified Chinese Characters, and for Japanese & Korean Characters. In fact, with the availability of the True Type font file in Pelutech, we can convert characters in any language to a bitmap image with minimal distortion.

Internal Code	Size	Font Color	Expected Result	Output Image
&H0041	18	Black	A	A
&H0041	16	Blue.	Z	Z
&H0041	14	Red	w	w
&H0065	16	Green	e	e

Table 5-2: Experimental Results of English Alphabets

Internal Code	Size	Font Color	Expected Result	Output Image
&H0031	18	Black	1	1
&H0039	16	Blue	9	9
&H0040	14	Red	@	@
&H0023	16	Green	#	#

Table 5-3: Experimental Results of Numbers and Symbols

Internal Code	Size	Font Color	Expected Result	Output Image
&H4E2D	18	Black	繁	繁
&H6587	16	Red	體	體
&H7535	18	Blue	电	电
&H8111	16	Black	脑	脑

Table 5-4: Experimental Results of Traditional and Simplified Chinese Characters

Internal Code	Size	Font Color	Expected Result	Output Image
&H30D3	18	Black	ビ	ビ
&H30B8	16	Red	ジ	ジ
&HC815	18	Blue	정	정
&HBCF4	16	Black	보	보

Table 5-5: Experimental Results of Japanese and Korean Characters

6. Rebuilding Form Controls

Many Web pages contain <FORM> controls that are used to gather information from users. However, the text inside the controls is normally rendered by browsers using local fonts because the HTML 4.01 specification does not support the <FORM> controls with any embedded image items for text objects [45]. Therefore, our system must regenerate and build simulated forms to avoid invoking local fonts, and make it free from CJK fonts. The Form Controls Builder module is designed to do that.

6.1 Elements of the Form Tags

Usually, when a browser uses GET to obtain a Web page, it first parses and analyzes the HTML code, it then generates the form control on the client side and display the font for the text items in the controls on the client side based on the information from the <FORM> tag. In the HTML 4.01 specification, there are ten different types of controls in the <INPUT> tag [46], as shown below:

```
<!ENTITY % InputType "(TEXT | PASSWORD | CHECKBOX | RADIO
| SUBMIT | RESET | FILE | HIDDEN | IMAGE | BUTTON)">
```

These ten controls play different roles in form handling in a Web page. The functions of each control are listed in Table 6-1 below:

Input Type	Function
TEXT	Creates a single-line text input control.
PASSWORD	Creates input text that is rendered to hide the actual characters (usually substituting them for a single character). This control type is often used for sensitive input such as passwords. Note that the current value is the text entered by the user, not the text rendered by the user agent.
CHECKBOX	Creates a checkbox.
RADIO	Creates a radio button.
SUBMIT	Creates a submit button that triggers the form action.
IMAGE	Creates a graphical submit button. The value of the "src" attribute specifies the URI of the image that will decorate the button.
RESET	Creates a reset button that resets the form value to initial.
BUTTON	Creates a push button. User agents should use the value of the value attribute as the button's label.
HIDDEN	Creates a hidden control to store form values.
FILE	Creates a file select control. User agents may use the value of the attribute as the initial file name.

Table 6-1: Functionality of each input type

In terms of their functionality, the ten controls can be further divided into four categories. The first category contains text elements displayed outside the controls such as the checkbox and radio controls. The second category needs to embed text into the control; for example, submit, reset and button controls. The third category includes text and password controls, which contain text fields and require the user to input text. The fourth category contains neither text nor text input by the user, and includes elements such as image, hidden and file controls. A summary of the controls is listed below in Table 6-2.

Category	Controls	Description
1	CHECKBOX, RADIO	Text elements are displayed outside the control
2	SUBMIT, RESET, BUTTON	Text elements are embedded in the control in the display
3	TEXT, PASSWORD	Text input is required from the user
4	IMAGE, HIDDEN, FILE	No text elements need to be displayed or input by the user

Table 6-2: Summary of the controls

Since category four controls do not have any text elements for display and they do not need any user input, they do not need any special handling by our gateway. Thus we only need to take care of the seven controls listed under the first three categories.

6.2 Handling of Checkbox and Radio Controls

The first category includes checkbox and radio controls in which text is separated from the controls. In fact, these controls support graphic display. Thus, the only work we need to do is: (1) obtain the text data, (2) convert the text to character images using the Character Image Constructor, (3) replace the text with images, and (4) send the images to the client browser to generate the control at the client side.

Figure 6-1 and Figure 6-2 show an example of an original check box control and its converted image generated by our Form Control Builder, respectively.

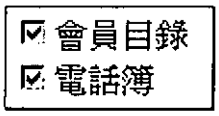


Figure 6-1: Original Checkbox Control

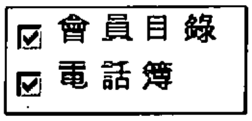


Figure 6-2: Simulated Checkbox Control generated by Pelutech

The HTML source code for the original checkbox control shown in Figure 6-1 is:

```
<input type="checkbox" checked>會員目錄<br>
<input type="checkbox" checked>電話簿<br>
```

The HTML source code for the simulated checkbox control that was generated by Pelutech is:

```
<input type="checkbox" checked>
  <img src="fc?f=6703"><img src="fc?f=54E1">
  <img src="fc?f=76EE"><img src="fc?f=9304"><br>
<input type="checkbox" checked>
  <img src="fc?f=96FB"><img src="fc?f=8A71">
  <img src="fc?f=7C3F"><br>
```

The only difference is that the text “會員目錄” and “電話簿” are displayed using locally installed fonts at the client machine in Figure 6-1, whereas the same text is displayed by character images in Figure 6-2 where no locally installed fonts are needed.

Figure 6-3 and Figure 6-4 show an example of an original radio control and its converted image based on the control generated by our Form Control Builder, respectively.

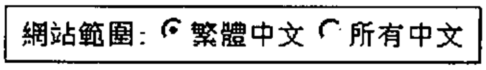


Figure 6-3: Original Radio Control and original HTML code

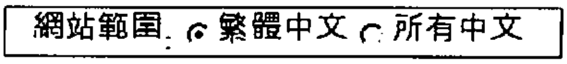


Figure 6-4: Simulated Radio Control and HTML code generated by Pelutech

The text in Figure 6-3 is displayed using a locally installed font but character images are used for display of the same text in Figure 6-4. The HTML source code of the original control is:

```
網站範圍:
<input type="radio" checked>繁體中文
<input type="radio">所有中文
```

The HTML source code for the simulated radio control generated by Pelutech is:

```
  <img src="fc?f=7DB2"><img src="fc?f=7AD9">
  <img src="fc?f=7BC4"><img src="fc?f=570D">
  <img src="fc?f=003A">
<input type="radio" checked>
  <img src="fc?f=7E41"><img src="fc?f=9AD4">
  <img src="fc?f=4E2D"><img src="fc?f=6587">
<input type="radio">
  <img src="fc?f=6240"><img src="fc?f=6709">
  <img src="fc?f=4E2D"><img src="fc?f=6587">
```

6.3 Handling of Submit, Reset and Button controls

The second category of form control in which text is embedded includes submit, reset and button controls. After the HTML Document Parser obtains the form name and the text from the VALUE attribute in the <INPUT> tag, the Character Image Constructor generates the CJK character images. Then, the Form Controls Builder constructs simulated control block images. Consider the following example:

```
<FORM NAME = "FormName" ACTION = "FORM.CGI">
  <INPUT TYPE = "SUBMIT" VALUE = "檢索">
</FORM>
```

First, the value "檢索" is retrieved from the tag by the HTML Document Parser. Then, the images "檢" and "索" are generated by the Character Image Constructor. Finally, the Form Controls Builder will embed the above character images into a blank button image " " to form the finalized image "

".

As the attribute of this control is changed from a button to an image, the type of the <INPUT> tag needs to be modified. In addition, the rebuilt images must also specify the tag so that the images can be correctly found by the client browser and displayed on the client machine. Consequently, the <INPUT> tag is modified as follows:

```
<FORM NAME = "FormName" ACTION = "FORM.CGI">
  <INPUT TYPE = "IMAGE" SRC = "FormBuilder?text=檢索"
  VALUE = "檢索">
</FORM>
```

Since the default action for the **IMAGE** type <INPUT> control is **SUBMIT**, the above method only works in the **SUBMIT** and **BUTTON** types of <INPUT> controls. It cannot be used to process **RESET** type controls. Instead, we use the link tag <A> and a JavaScript function to trigger the reset action when a user clicks the reset button. Consider the following example:

```
<FORM NAME = "FormName" ACTION = "FORM.CGI">
  <INPUT TYPE = "RESET" VALUE = "重設">
</FORM>
```

The form "FormName" contains a reset button to reset the form value to the initial state. After the Form Controls Builder constructs a simulated reset control image, the <INPUT> tag is replaced by a link and an image tag as shown below.

```
<A HREF = "javascript:document.FormName.reset( )">
    <IMG SRC = "FormBuilder?text=重設">
</A>
```

Note that the function "javascript:document.FormName.reset( )" is used to reset the Form to its initial value in the HTML document.

The solutions provided in our Form Control Builder can not only generate the same look and feel of the form control display, but also provide the same functionality as the original controls. Figure 6-5 shows an original button control and Figure 6-6 a simulated button control.

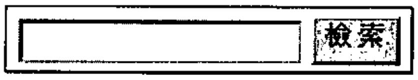


Figure 6-5: Original Button Control

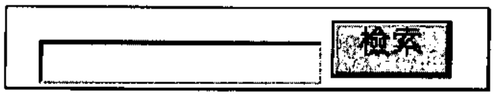


Figure 6-6: Simulated Button Control generated by Pelutech

However, there are some limitations in Pelutech. The current system cannot handle certain *ill-formatted* HTML forms. For example, a well-formatted text control form should be coded as:

```
<input type= "text" size=20 name="Text01">
```

An ill-formatted HTML form as quoted directly from the Web site "Yahoo! Chinese" might look like this:

```
<input size=20 name="Text01">
```

Pelutech cannot handle this code because our HTML parser has difficulty identifying what type of control this "INPUT" tag belongs to. The only way to solve this problem at present is to use some predefined rules in the HTML parser.

6.4 Handling of Text and Password Controls

The third category of form control includes text and password controls that require text input from users. Since these controls require input from a client, some substitute input method must be provided. In this section, we will discuss how to use the Input Method Engine through the Input Method Interface. The design of the Input Method Engine will be discussed in Chapter 7.

When the HTML Document Parser detects the text field in a form, such as the code in the following example:

```
Original code:
<FORM NAME = "FormName" ACTION = "FORM.CGI">
  <INPUT TYPE = "TEXT" NAME = "TextField">
</FORM>
```

The Form Controls Builder will insert the Input Method Interface to establish a connection with the Input Method Engine as shown below:

```
Modified code for the Form:
<FORM NAME = "FormName" ACTION = "FORM.CGI">
  <INPUT  TYPE  =  "TEXT"  NAME  =  "TextField"  onfocus  =
  "SetIMEfocus(FormName.TextField)">
</FORM>
```

For this implementation, the JavaScript built-in function *onfocus* is inserted into the <INPUT> tag. The *SetIMEfocus* is an embedded function inserted into the modified form to establish a connection with the Input Method Engine when the user clicks on that field. Finally, the modified control is sent to the client side, and will be rendered by the Internet browser.

7. Native Input Method Engine

Without localized input methods installed, applications cannot accept non-English input from clients. Generally, users are expected to find their own methods to enter CJK characters. However, in Pelutech we do not assume that a client has a proper CJK platform. Thus, we cannot assume that the client has a properly installed local input method engine. For this reason, Pelutech also provides an Input Method Engine (IME). The primary purpose of this IME is to facilitate input required by certain interactive Web pages using CJK characters. IME is also needed to facilitate search for CJK characters in the reformatted Web pages.

7.1 Input Method Engine Design

Input methods are software components that make use of the operations on the input device to produce text input for applications. In Pelutech, since we do not assume there is a locally installed native input method engine, we need to provide that function through the Pelutech Input Method Engine (PIME). PIME also has an input method interface, which is delivered to the client side with the Pelutech reformatted Web page. This interface module runs on the client machine and communicates with PIME to deliver input method services to clients. The design of PIME is based on the Input Method Framework defined in the Java 2 Specification [47, 48]. It allows users to input CJK text without installing any third-party Input Method Engine in the client machines.

The framework architecture is modeled as an Object Oriented system. Components of the framework communicate with each other only through a well-defined interface. The internal structure and implementation of each component is hidden and independent of the other components. Each independent input method is also defined as a component with a well-defined interface so that it can be plugged into the IME and used by other components [49].

This Input Method Framework consists of two major components: the Native Interface and the Core Engine. The *Native Interface* provides an interface for a client application, which is called the Application Programming Interface (API). This API is responsible for handling the communication between user input in a text input component and the Core Engine, which provides the different input methods. The *Core Engine* supplies the different input methods and it has a Service Provider Interface (SPI) for flexibility in allowing different input methods to be plugged into the engine.

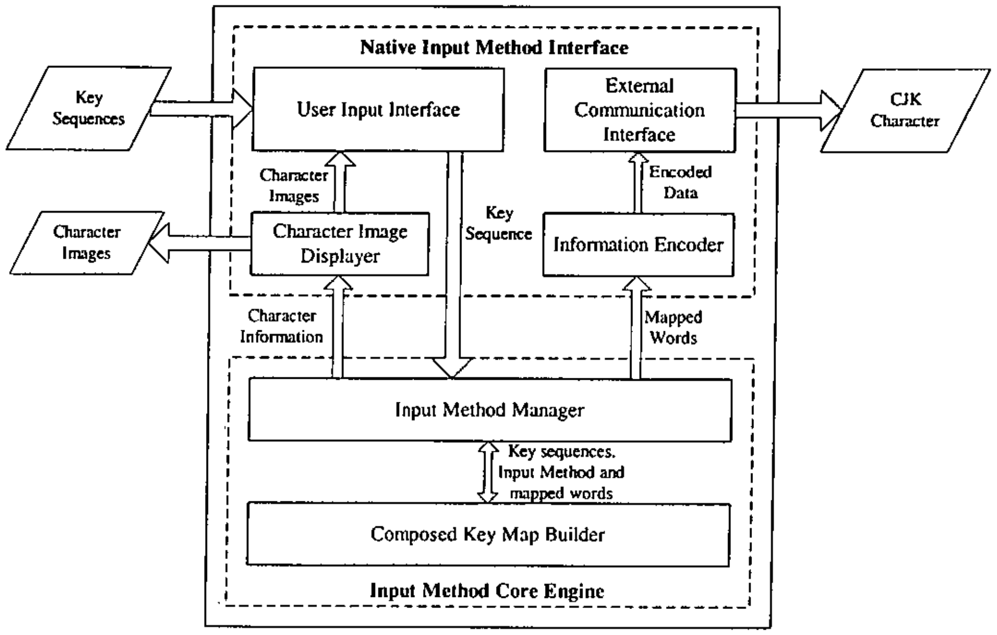


Figure 7-1: Native Input Method Engine Framework

The division of the Native Interface and the Core Engine gives more flexibility for the implementation of the input method services. The Core Engine can either be installed at the server-side or the client-side and transparent to the client. Figure 7-1 shows the major components of the input method engine. Details of each of the sub-components are explained in the subsequent sections.

7.1.1 Native Interface

The Native Interface provides the input method interface to the client and it also communicates with the Core Engine on behalf of the client to obtain the right input method, in the chosen style. It supports two service levels: the Integrated Text Input User Interface and the Non-Integrated Text Input User Interface. Each of these service levels has a separate style of input method, which is summarized in Table 7-1.

Service Level	Input Style	Details of the style
Integrated Text Input User Interface	On-the-spot	When composing the text, the current candidate is highlighted. If a user chooses a different candidate, the new one replaces the previous candidate text.
	Below-the-spot	The composed text window is close to the insertion point of the text components. The text is inserted after the insertion point after the composed text has been committed.
Non-Integrated Text Input Interface	Root-window	A separate composition window is prepared to accept the composed text. Then the committed text is sent to the application. Unlike the Integrated Text Input Interface, users are required to place the insertion point.

Table 7-1: Service Levels and Input Style supported by input method client

As shown in Figure 7-1, the Input Method Interface component is composed of four sub-components: the User Input Interface, Character Image Displayer, Information Encoder and External Communication Interface. The roles of these sub-components act like a bridge between the Input Method Engine and the external working environment, such as a Web browser or a Java Applet.

The *User Input Interface* appears as a text field to receive keyboard input from the user directly. It provides basic editing functionality so that user can modify text. It is also connected to the Character Image Displayer to display retrieved character images.

The *Character Image Displayer* is responsible for rendering the character image in the User Interface since local fonts are unavailable. It is connected to the User Input Interface and the Input Method Manager. When it receives character code from these two (connected) components, it will look for the correct character glyph and generate the character image. Since there is a Character Image Constructor inside Pelutech, the major function of the Character Image Displayer is passing information from the Input Method Engine to the Character Image Constructor. The Character Image Displayer can be a separate component when the Input Method Engine is a stand-alone application.

The *External Communication Interface* communicates with the Web browser and the Core Engine of Pelutech. It receives data from the Input Method Engine in its internal format and converts it to a format that the external environment can understand. This internal data includes character codes in the text box and the current character encoding information. Therefore, the external environment can obtain the necessary information from this interface for further processing, such as searching or submitting a form.

The *Information Encoder* is used to convert the character encoding and related information into a standardized format. This standardized format ensures that the information is transmitted correctly via the Internet and is understood by other applications. This conversion is used because some special characters and double-byte characters (especially CJK characters) need to follow an "escape scheme" before being transmitted via the Internet. For example, a punctuation mark cannot appear in the Uniform Resource Locator (URL) string, so a string is converted into a MIME format called the "x-www-form-urlencoded" format [45]. According to these schemes, the Information Encoder converts the data into various formats in order to conform to standards that are based on the following rules.

- The ASCII characters 'a' through 'z', 'A' through 'Z', and '0' through '9' remain the same.
- The space character ' ' is converted into a plus sign '+'.
- All other characters are converted into the 3-character string "%xy", where xy is the two-digit hexadecimal representation of the lower 8-bits of the character.

Besides this, character codes of the user text are also converted to hexadecimal numbers before they are sent through the interface. For example, the Traditional Chinese Big5 code for character one is represented by A440. This prevents misinterpretation of any of the double-byte characters as two separate ASCII characters.

7.1.2 The Core Engine

The Core Engine is responsible for processing the user input and generating the text output according to a specified input method. It comprises two main sub-components: the Input Method Manager and the Composed Key Map Builder. The Input Method Manager manages different input methods and searches for a character according to the user input data.

The *Input Method Manager* is the core of the Core Engine. After it receives a composed key sequence from its user interface, it maps this key sequence into an actual character based on the user selected input method.

The *Composed Key Map Builder* locates the input method resource and builds the input method mapping data structure for the Input Method Manager search. The Input Method Manager informs the key map builder that a specific input method is selected, and then it reads the input method mapping data stored at the Pelutech server before it builds the mapping table at the client side.

7.2 System Implementation

While the Input Method Manager manages various input methods, the Composed Key Map Builder builds the composed key-mapping table. After an input method has been specified by the user, the Input Method Manager calls the Composed Key Map Builder to build the corresponding mapping table. The Composed Key Map Builder searches for the mapping table resource at the Pelutech server. The mapping table resource format is the same as the Textual Input Table (.tit) format used by the CXTerm for UNIX's X Windows system [50]. The Textual Input Table file can be downloaded from the Internet, so that the input method can be updated easily. It allows new or additional input methods to be added later.

7.2.1 Composed Key Retrieval Methodology

After reading the Textual Input Table, the Composed Key Map Builder stores the map in memory. There are two approaches for the retrieval of composed key sequence: using a data structure or a database.

7.2.1.1 Retrieve Key Sequence using Data Structure

With the aim of fast retrieval of the character from the composed key sequence, a special data structure and searching algorithm should be designed. The *Hash Table* and *Trie Data Structure* were studied during the implementation of the input method engine.

Although the hash table can look for characters by submitting the composed key sequence, it can only perform exact matching. For example, the composed key sequence "P" of the Changjie input method (倉頡) is only mapped to a Chinese word "心" but it cannot display other words that start with the same prefix, such as the Chinese word "忙" with a "PVF" composed key sequence. However, in the Internet world, it is better to provide some hints for users to input text because they may not be able to consult a reference manual for the composed key sequence.

Instead of searching by comparison through the entire composed keys sequence, each of the characters in the key sequence is used to determine a multi-way branch at each step of the trie data structure.

Figure 7-2 shows the Trie tree of the Changjie input method. When the user enters a composed key character, such as "A", the searching function (such as trie class's *searchTrie()* method in Java) will traverse the tree. From the figure, it will find characters "日昌明" because they have the same composed key sequence prefix "A".

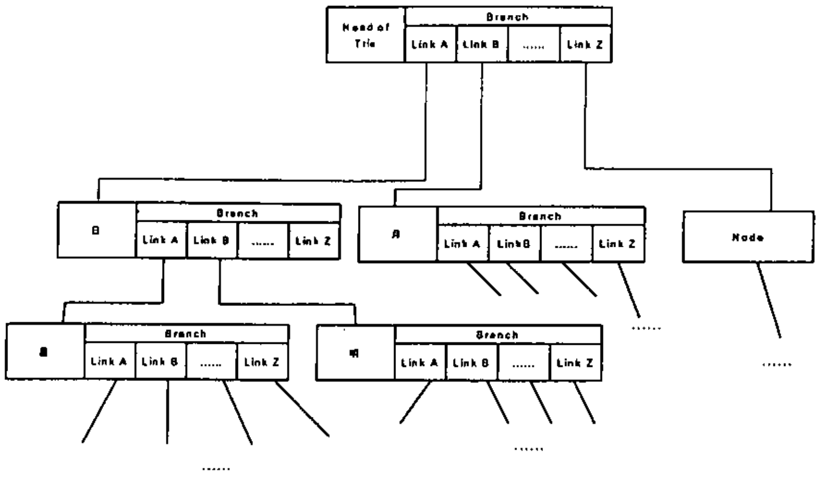


Figure 7-2: Trie Tree of the Changjie input method

A trie tree is also an efficient algorithm for building the mapping table. From Robert L. KRUSE [51]:

"The number of steps required to search a Trie (or insert into it) is proportional to the number of characters making up a key, not to a logarithm of the number of keys, as in other tree-based searches."

Since the maximum number of characters in a composed key sequence is often 5, the number of characters in a sequence is small and it is faster than a binary tree search. In 5 iterations, it can locate 265 (11,881,376) keys.

The trie data structure was chosen to build the input method mapping table (in memory) due to certain advantages, which are described above.

7.2.1.2 Retrieve Key Sequence using Database

Another approach for storing the composed key sequence is to use a database. First, the key sequence mapping table is converted into a database table. For example, the table TABLE_CJ is created for the mapping table of Changjie. There are two columns in this table: KEY_SEQ is used to store the key sequences; and WORDS is used to store words that map from the key sequences. When a user inputs a key sequence, the system retrieves the matching word using a SQL statement like the following (for a sequence beginning with "A"):

```
select WORDS
from TABLE_CJ
where KEY_SEQ like 'A?'
```

However, if the entire mapping table is inserted into a database table, the retrieval time for each query takes too much time. Therefore, for faster retrieval of characters from the composed key sequence, we create 26 materialized views [52] that store the key sequence that begins A to Z separately. For example, the materialized view of key sequences that begin with A for Changjie is:

```
create materialized view VIEW_CJ_A
from TABLE_CJ
where KET_SEQ like 'A_'
```

By employing this method, the retrieval time for each key sequence can be reduced to 1/26th of the time using the original naïve method. For example, when a user inputs the key sequence "A" , then the system can find characters "日昌明" from the view VIEW_CJ_A, and there is no need to use table TABLE_CJ. The SQL statement for retrieving matching words for key sequences that begin with "A" is as follows:

```
Select WORDS
from VIEW_CJ_A
where KET_SEQ like 'A?'
```

7.2.2 System Implementation

Since the Native Engine is mainly designed for entering CJK text in the Web environment, a Web-based programming language is chosen to implement the Input Method Engine. In this thesis, we implement the input method engine in two ways: Client-side Input Method Engine and Server-side Input Method Engine. The Client-side Input Method Engine is designed as a Java Applet and the trie tree and hash table are used to retrieve key sequences. For the Server-side Input Method Engine, JavaScript is used together with Active Server Page (ASP) to connect to the back-end database.

7.2.2.1 Client-side Input Method Engine

This Input Method Engine is implemented as a Java applet in a Java Input Method Engine [53]. The Input Method Engine is downloaded with the reformatted Web page to the client side. The Java Input Method Engine applet currently supports the following input methods:

(i) Changjie Input Method for the Big 5 Chinese characters

(ii) Simplex Input Method for the Big 5 Chinese characters.

The User Input Interface provides a text field for the user to enter text. In fact, it is not a regular text field; instead it is the graphically painted canvas of the downloaded Java Input Method Engine applet. A rectangular text field and the character images are drawn in the canvas. Moreover, the applet implements the *KeyListener* method, so that whenever a user presses a key on the keyboard, it detects whether it is an action key or a normal character and then performs different functions depending on the keystroke information. An action key is used to perform a specified function, such as the backspace key that is defined to erase the previous entered characters in the text field. If it detects a normal character keystroke, it will pass the keystroke information to the Character Image Displayer to render the character image. If it detects a special combination of keystrokes such as [Ctrl] – [Alt] – [Tab], it will call an input method window that allows CJK text to be entered. A user can select the input method from the user interface as shown in the Figure 7-3.

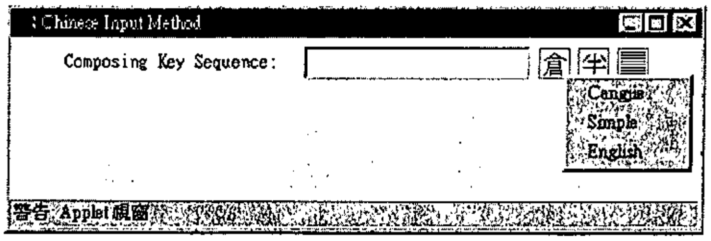


Figure 7-3: Java Input Method Engine

First, the Character Image Displayer makes a request to the server and then uses an instance of the *MediaTracker* class to wait for the image from the Pelutech server. The MediaTracker class is an Abstract Window Toolkit (AWT) class that keeps track of the status of media objects [54].

To allow communication between the Java Input Method Engine applet and the external environment, *Netscape LiveConnect Technology* is used to connect them together [55]. In this interface implementation, only JavaScript-to-Java communication is required. This kind of communication can be used to control the behavior of the Java applet through JavaScript [56]. To enjoy this feature, the method of the Java class is declared as public. Declaring the method as public method makes it externally visible so that it can be called from the JavaScript code. Currently, Netscape version 3.x (or above) and Internet Explorer 4.x (or above) support this JavaScript-to-Java communication. The following JavaScript and Java applet source code shows how the Netscape LiveConnect technology allows this connection.

```
<SCRIPT LANGUAGE=Javascript>
<!--
msgWindow = window.open("","displayWindow", "menubar=no,
scrollbars=no, status=no, height=10, width=500");
msgWindow.document.write("\""+ document.KeyApplet.getSrcText() + "\"");
//-->
</SCRIPT>
......
<APPLET name="KeyApplet" code="Key.class" width=200 height=30>Applet
Area</APPLET>
```

Figure 7-4: JavaScript Source

```
import java.lang.*;
...
public class Key extends Applet{
......
 public String getSrcText(){
   String srcText = null;
   ......
   return srcText;
 }
}
```

Figure 7-5: Java Applet Source

From the source code above, the embedded JavaScript calls a Java applet named KeyApplet, which is implemented by the class Key that has the *getSrcText()* method. When getSrcText() is called, it returns a string. The JavaScript code then writes out this string to a newly opened window called displayWindow.

In addition, since the Information Encoder is required to prepare the information to conform to a standard MIME format, the Java *URLEncoder* class is used to convert all the characters to a string [54].

In the Java Input Method Engine, a hash table and a trie tree are used in the Composed Key Map Builder. The Java API has a *Hashtable* class that maps keys to values. By using Hashtable's *put()* method, the composed key sequences and the corresponding characters can be put into a hash table.

Two Java classes are written to implement the trie data structure. They are the *Trie class* and the *Node class*. Their class relationship and their design are shown in Figure 7-6. The Trie class provides methods to build a trie tree, insert nodes and traverse the tree. The Trie tree is composed of Node instances. The Node class provides methods *getData()* and *setData()* to get characters from a node and store characters in a node, respectively. In addition, *getBranch()* and *setBranch()* methods are responsible for traversing and linking the nodes, respectively. Nodes are linked together by the trie tree. From another perspective, the branch of a trie tree is a linked list. This can save memory space because a node does not contain any characters and uses pointers. In addition, using a trie data structure groups together all the possible candidate characters with the same composed key sequence prefix.

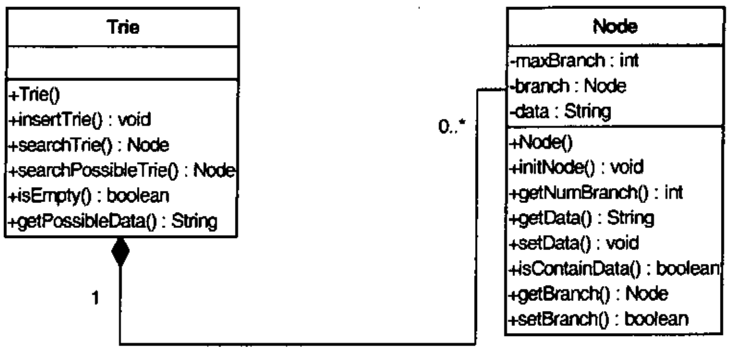
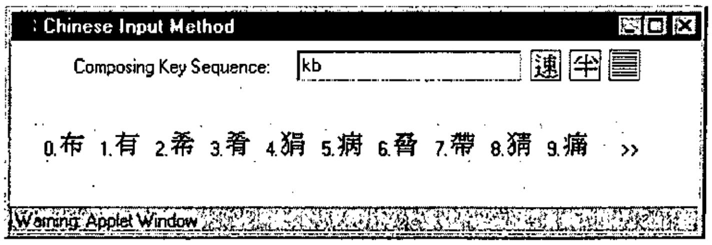


Figure 7-6: Class Diagram of Trie and Node

Finally, we perform an experiment on the Java Input Method Engine to not only tests its functionality and performance, but also to demonstrate how the image character text field listens for a keystroke.

When the composed key sequence is entered in the text field, it will display all the possible character candidates that have the same composed key sequence prefix, as shown in Figure 7-7.



kb

0.布 1.有 2.希 3.肴 4.狷 5.痏 6.胥 7.帶 8.猜 9.痛 >>

Figure 7-7: Same composed key sequence prefix displayed

From this experiment, it was found that although more composed key characters are entered into the text field, the time required to load the candidate characters is reduced. This is because the number of candidates with the same composed key sequence prefix becomes smaller as the composed key gets longer.

The size of the Input Method Engine applet is around 100 K Bytes. The loading speed experiment shows that it takes around 20 seconds to load the applet into the Web browser using a standard modem connection. If a broad bandwidth connection is used, such as 1.5M bits, it takes a negligible amount of time to load the applet. This experiment shows that the loading time of the applet is within acceptable limits for the slow network connection.

Besides this, it is found that Web browsers, Microsoft Internet Explorer version 5.x or Netscape Navigator version 4.x, may actually store the applet in a cache, and this dramatically improves subsequent loading time for the applet.

7.2.2.2 Server-side Input Method Engine

The Server-side Input Method Engine is developed in JavaScript and ASP and uses the database approach. The JavaScript code is used to pass the information between the original HTML document and the input method window that is coded in ASP. Then, ASP queries the database and processes the user's input.

To implement this method, first, we need to embed a JavaScript file in the HTML document header as follows:

```
<SCRIPT language="JavaScript1.2" SRC="enable.js">
</SCRIPT>
```

The purpose of this JavaScript file is to provide some useful functions that support passing the information between the original HTML document and the input method window. These functions ask as a bridge to provide a communication channel for passing the input text from the input method window back to the original form.

To activate the input method, a function *OpenIMEWindow()* is developed to open the input method window. There are two parameters in this function: the first parameter tells the input method engine which input method is to be used, and the second parameter declares the text field that is used to retrieve text from the input method engine. Therefore, the function not only opens the input method window, but also passes the text field that is used to retrieve the text from the input method. For example, if there is a form called "form1" and a text field "textfield1" that receives the text from the input method engine; then the following code needs to be added:

```
...
<form name="form1">
Field    1:    <input    name="textfield1"    onFocus    =
"IMEFocus('form1.textfield1');">
<input      type=button      value="CJ"      onClick      =
"OpenIMEWindow('Changjie', 'form1.textfield1');">
...
```

There is only one parameter in the function *IMEFocus()* because this function is only used to tell the input method engine which text field is to be used to receive the text input from the user.

When the user inputs a key sequence, the input method engine will query the database and then retrieve the corresponding characters and display them on the screen. Then, the user can press the keys "0" to "9" to select the character and select "Send to Form" after all the characters have been input. The sample screen for the Server-side Input Method Engine is shown in Figure 7-8.

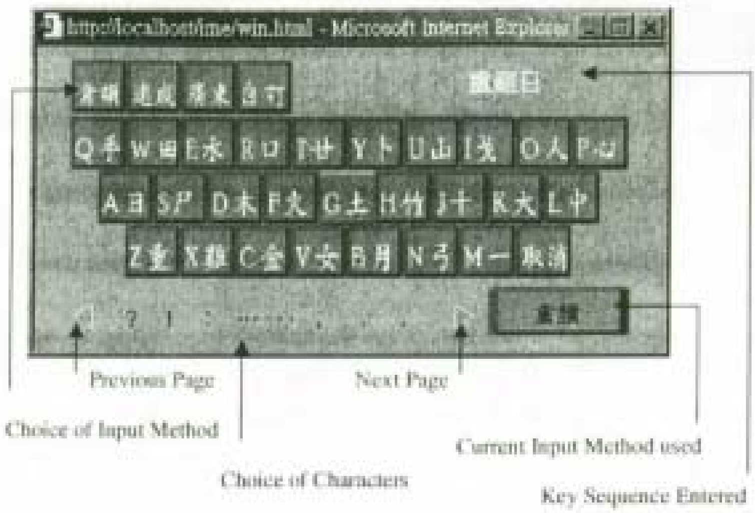

http://localhost/ime/win.html - Microsoft Internet Explorer
Q手 W田 E水 R口 T廿 Y卜 U山 I戈 O人 P心
A日 SP D木 F火 G土 H竹 J十 K大 L中
Z重 X難 C金 V女 B月 N弓 M一
Previous Page
Next Page
Choice of Input Method
Choice of Characters
Current Input Method used
Key Sequence Entered

Figure 7-8: Server-side Input Method Engine

7.2.3 Discussion

The two implementations (described above) provide different approaches for the development of the Input Method Engine. These Input Method Engines not only let a user input text without installing the Input Method Engine locally, but also provide a cross platform input method framework for developer. However, these Input Method Engines contain some weaknesses.

In the database approach, the server needs to query the database when a user inputs a key sequence. The utilization of the server will be quite high when there are many concurrent users.

In comparison, the Client-side Input Method Engine only provides complete support for both the input method client API and the input method engine SPI when the Java Software Development Kit (JSDK) from version 1.3 onwards is used. An older version of JSDK cannot be used on the client, and an older version for the engine means losing the flexibility of the SPI. It should be noted that, the two dominant browsers, Internet Explorer and Netscape, are most often equipped with the old version of the Java Virtual Machine (JVM), which only supports the Java Software Development Kit Version 1.1 or older. Users may need to download a Web browser software patch to update the JVM, or they need to install the latest Java Runtime Environment version 1.3. This installation work is not an easy task for a normal user since it involves system configuration and administration work. These backward compatibility issues should be considered when designing a new input method.

In addition, although Java is designed to fully support the Unicode strings and Internationalization (I18N), it obviously still needs the Unicode fonts to be installed in the target platform to display the text properly [57]. I18N is a software development technique that separates the language elements from the program logic, so that the program can run in any language environment without redesign. Java has already provided APIs to support I18N, such as the API for processing locale, numbers, currency and date [58]. These APIs are designed for Unicode 2.0. However, the current version cannot display CJK characters without the CJK fonts being installed. From the Java Developer Connection chat forum, it was pointed out that CJK fonts would be made available for developers later [59].

8. Character Image Search Engine

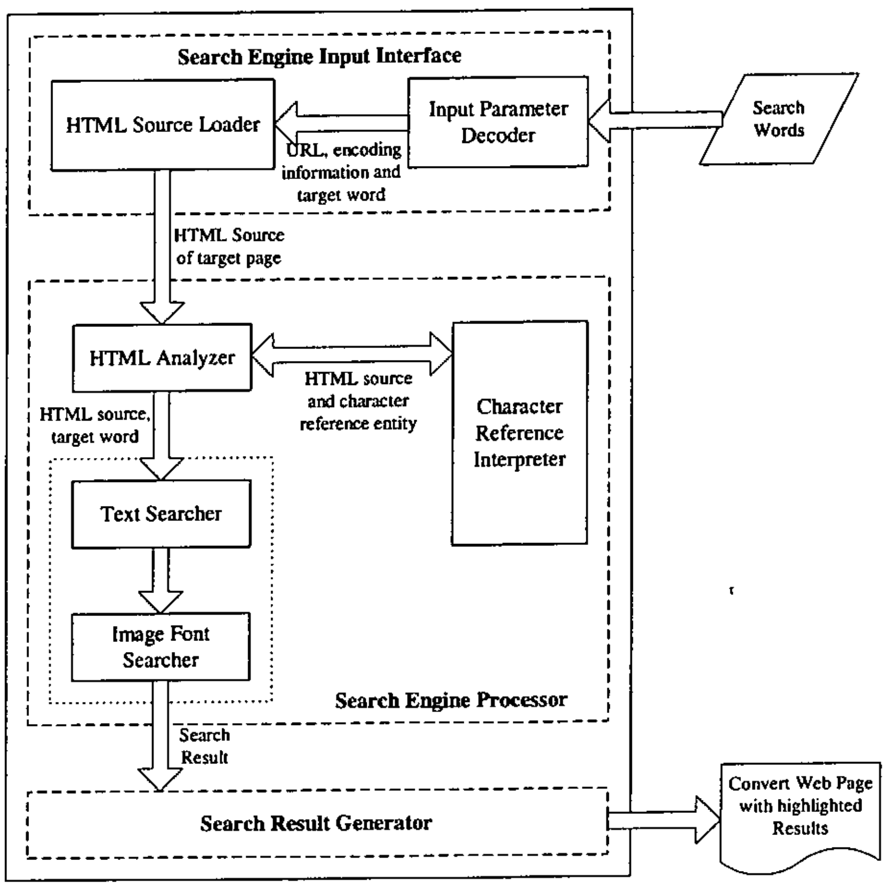
Most Web browsers provide functions for content search. Sometimes, when given a large Web page, the user may want to look for a specific topic or word in that page. The content search is designed for text search in the current page and it makes browsing much easier. The search function works very much the same way as in any word processor where the words found are highlighted. It is well known that the search functions can be applied only to text content.

Therefore, Web browser search functions cannot find CJK characters in the reformatted Web pages from Pelutech because CJK characters are now converted to character images. Their content type is also changed because all CJK characters are labeled in HTML by the <IMG> image tag. Character-to-Image replacement without the function for character search would make such schemes less useful. To maintain the same search functionality, the Character Image Search Engine is designed in Pelutech to overcome this limitation.

It should be noted that the reformatted Web pages might be mixed with text and character images because the Conversion Engine in Pelutech will not convert all the characters in a Web page if the client already has the corresponding font. As a result, this Character Image Search Engine must be capable of performing both text search and character image search and also mixed-mode search. In addition, words found as a result of a search should also be highlighted to give visual cues to the user.

8.1 Design of Character Image Search Engine

The Character Image Search Engine (CISE) comprises three parts: the Input Interface, the Searching Processor and the Result Generator. The major components of the CISE are shown in Figure 8-1.



Search Engine Input Interface
HTML Source Loader
Input Parameter Decoder
URL, encoding information and target word
Search Words
HTML Source of target page
HTML Analyzer
HTML source and character reference entity
Character Reference Interpreter
HTML source, target word
Text Searcher
Image Font Searcher
Search Engine Processor
Search Result
Search Result Generator
Convert Web Page with highlighted Results

Figure 8-1: Components of the Character Image Search Engine and their relationship

The Input Interface is responsible for retrieving necessary information from the user, so it is located at the client side. It consists of a Native Input Method Engine and an Input Parameter Decoder. The *Native Input Method Engine*, which was mentioned in the previous chapter, is used to gather the user input. The Input Parameter Decoder decodes those parameters passed by the client browser, such as the current URL address, encoding support and the word that is the target of the search.

The *Searching Processor* is the key component of the CISE. It looks for the words given in the Input Interface. The Searching Processor is composed of an HTML Analyzer, a Character Reference Interpreter, a Text Searcher and a Character Image Searcher. The HTML Analyzer is responsible for analyzing the HTML tags and content. The Character Reference Interpreter interprets the character reference entity for use in the HTML Analyzer. The Text Searcher and the Character Image Searcher are responsible for text search and character image search, respectively. Unlike the Input Interface, the Searching Processor can be located at either the client-side or the server-side. When it is located at the server-side, the search can be performed using the original HTML source page; otherwise, at the client side it can only perform searching in the converted page.

The *Search Result Generator* is responsible for presenting the word found in the Web page. If the word is found, it will highlight the word found. Otherwise, it will report to the user that the word is not found.

We will first talk about the components inside the Input Interface and the Searching Processor, and the searching approaches will be discussed in Section 8.2.

8.1.1 Input Parameter Decoder

The Input Parameter Decoder is responsible for decoding the input parameters. The input parameters are often encoded in "x-www-form-urlencoded" format when using a GET HTTP Request. Special characters and punctuation are encoded. Therefore, the Input Parameter Decoder has to decode this. The input parameters are the URL address, the current encoding, and the word that is target of the search. To prevent any misinterpretation of the double-byte CJK characters during transmission, the Search Engine Input Interface accepts search words in a hexadecimal string format. As a result, the Input Parameter Decoder needs to convert the hexadecimal string back into the original characters for further processing.

8.1.2 HTML Analyzer

The HTML Analyzer is responsible for analyzing the source HTML structure. It parses the entire HTML source code and distinguishes between different HTML tags in the source code. Then, it builds up a linked list data structure according to the parsed result. From the attribute of the node in the linked list, it can tell whether the tag is or is not a character image.

8.1.3 Character Reference Interpreter

The Character Reference Interpreter is responsible for interpreting the character entity references in HTML. It should be noted that, HTML character entity references appear in two forms: Numeric Character References and Character Entity References [45].

Numeric Character References specify the code position for a character in the document character set. For example, the numeric reference &#229 (in decimal) represents "å". Instead of using a meaningless numeric code to represent a character, HTML offers a set of character entity references. For example, "&lt;" is used to represent the < (less than) sign.

Therefore, the Character Reference Interpreter converts these references back into the actual numeric reference codes.

8.1.4 Text Searcher

The Text Searcher is responsible for text searching. It provides text search function like the Web browser's search function. It skips those special tag elements from the HTML source code, such as <SCRIPT>, <OPTION>, <HEAD>, etc.

8.1.5 Character Image Searcher

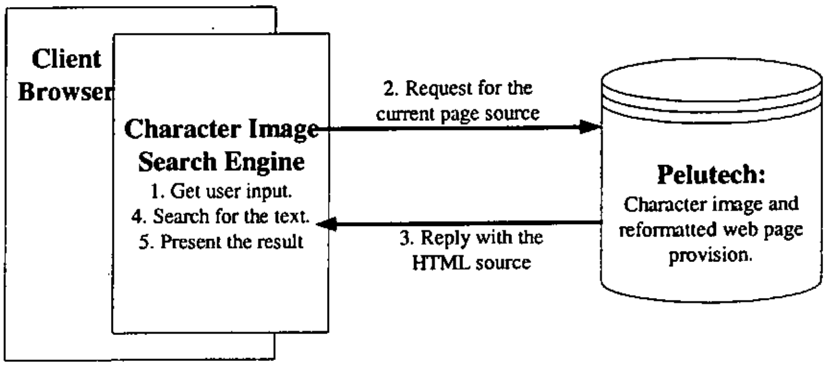
The Character Image Searcher is responsible for character image searching. It is based on a tag-pattern matching technique. It searches for the IMAGE tag of the corresponding text. To facilitate the image character search, the Conversion Engine in Pelutech provides a character-image indexing scheme. It uses the following scheme to represent characters: the Big 5 Traditional Chinese character "一" has an IMG tag <img src= "http://Pelutech/f?A440" width=16 height=16 …… alt = "一" >. The image name A440 is the Big5 code for "一" . Therefore, the CJK character can be indexed using the actual image name.

Therefore, the Character Image Searcher is required to handle CJK character images and English word images differently. Firstly, the character image searcher scans the entire source and looks for the character image's IMG tag produced by Pelutech. Then, it checks whether the words that is going to try and find contain an English word token or CJK characters. It will search for the word according to the corresponding IMG tag pattern. For example, if it searches for character "一" , it will try to match the tag pattern <img src= "http://Pelutech/f?A440" in the HTML source, and then notify the Search Result Generator to highlight the image.

8.2 Implementation of the Character Image Search Engine

The CISE is mainly designed for text searching and character image searching by Pelutech. There are two approaches to implement the Character Image Search Engine: the client-side approach and the server-side approach.

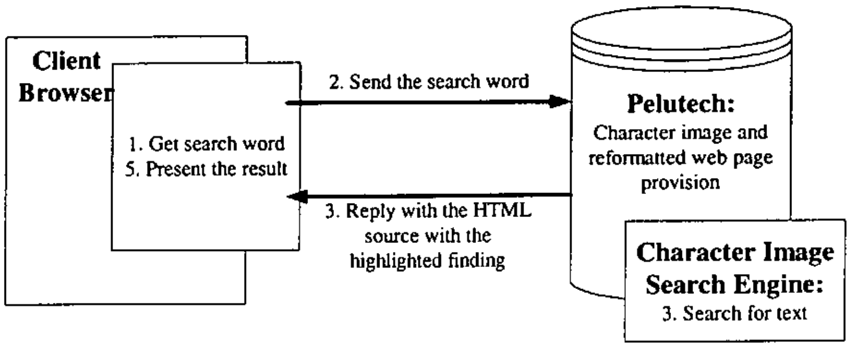
In the client-side approach, CISE is provided with a re-formatted Web page and executes on the client side host machine. Whenever there is a search request, it will do all the processing on the client host machine. This is shown in the Figure 8-2.



Client Browser
Character Image Search Engine
1. Get user input.
4. Search for the text.
5. Present the result
2. Request for the current page source
3. Reply with the HTML source
Pelutech:
Character image and reformatted web page provision.

Figure 8-2: Client-side Implementation of Character Image Search Engine

In the server-side approach, the search engine runs on the Pelutech server. Client search requests are sent to the server program through the Internet. This is illustrated in the Figure 8-3.



Client Browser
1. Get search word
5. Present the result
2. Send the search word
3. Reply with the HTML source with the highlighted finding
Pelutech:
Character image and reformatted web page provision
Character Image Search Engine:
3. Search for text

Figure 8-3: Server-side Implementation of Character Image Search Engine

8.2.1 Client-side Approach

In the client-side approach, CISE is integrated with the Input Method Engine. To integrate seamlessly in this implementation, the Java Input Method Engine is embedded into the search engine and the Java programming language is used for implementation.

Based on the search engine design mentioned in Section 8-1, it has to retrieve the current HTML source so that it can perform the search operation. However, the Java applet cannot access the current source Web page stored at the local cache due to security issues [60]. Java applets can only access resources from its originating server [61]. Therefore, the search engine applet makes an HTTP connection to the Pelutech Server to obtain the current HTML source. The applet then originates from the Pelutech server; otherwise, it cannot access the remote resources.

First, the CISE obtains the current URL address from the Web browser environment. This can be done using the LiveConnect Technology [55]. A section of JavaScript code embedded in the reformatted Web page gets the current URL address and then passes that address to the Java applet. Using Netscape LiveConnect technology, JavaScript can pass parameters to a Java applet. After getting the address from the JavaScript, the applet makes a URL connection to the Pelutech server and retrieves the HTML source.

The Java URL class is used to implement the HTML Loader. Using URL instance method *openStream()*, it opens a connection to the given URL and then uses *InputStream()* to read from that connection. Using this URL class, it is possible to cache the retrieved file content. Therefore, after the Web content has been stored in the cache the first time, the same Web page access will be loaded from the cache to reduce any subsequent access times.

With the HTML source, the Search Processor of the Character Image Search Engine can search for the text in the HTML source at the client side. It performs both text search and character image search.

In addition, the *HTMLList* and *HTMLData* classes are designed to manipulate and store the parsed HTML information. Indeed, HTMLList is the actual implementation of the HTML Analyzer. The class diagram is shown in Figure 8-4. HTMLList is a linked list data structure that stores the HTML source in an organized manner. Its parse method determines four types of HTML elements: the HTML element tag, element content, character reference entity and special element tag (e.g., <SCRIPT>, <OPTION>, <AREA>). This classification scheme helps the HTMLList to look for words in the HTML source systematically and efficiently.

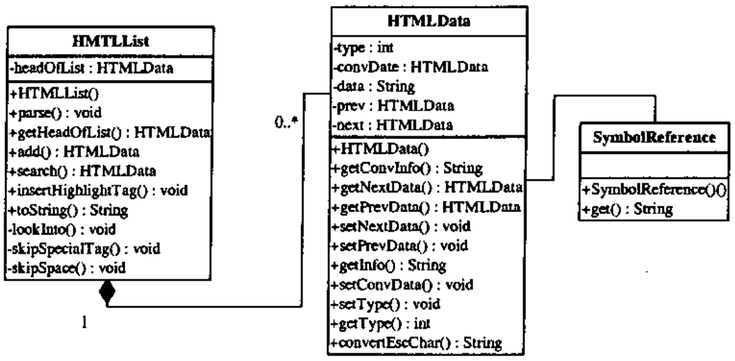


Figure 8-4: Class Diagram of HTMLList and HTMLData

For example, if the HTML source code is like that:

```
<HTML>
  <BODY>&lt;abc</BODY>
</HTML>
```

It can parse the HTML code and store it in a linked list. An example of the structure of the parsed HTML is shown in Table 8-1. Each table entry represents an HTMLData instance. Class attributes "Data" and "convData" store the original HTML token and the actual character that is represented by the character entity reference, respectively.

Data	convData	type
<HTML>	NULL	0
<HEAD></HEAD>	NULL	4
<BODY>	NULL	0
&lt;	"<"	2
A	NULL	1
...	...	...

* Notes:

Type 0: Normal HTML Text Tag

Type 1: Tag Element content

Type 2: Reference Entity

Type 3: Highlight Tag

Type 4: Special Tag which needs to skip when searching

Table 8-1: Parsed HTML code's data structure in the HTMLList

The *SymbolReference* class stores the complete list of the character entity references defined in HTML 4.01 [35]. The *convEscChar()* method in HTMLData can refer to this class when it needs to convert the character entity references into the actual character.

Highlighting tags are used to highlight the search result. Cascading Style Sheet (CSS) technology is employed for this purpose. For example, if a user looks for a string "Polytechnic" in a sentence "Hong Kong Polytechnic University", then the result will be highlighted using the following HTML code:

```
Hong Kong <b style = "background-image:url(yellow.gif);">
Polytechnic</b> University.
```

Using a linked list like structure in HTMLList, means that the highlighting tag can be inserted into the source HTML efficiently because the Character Image Search Engine only needs to find the insertion position and manipulate the object references.

The role of the Java Input Method Engine is to obtain the target search word from the user and then trigger the Character Image Search Engine.

8.2.2 Server-side Approach

The CISE is embedded into Pelutech when the server-side approach is used. As shown in Figure 8-3 in the previous section, the search engine is executed in the Pelutech server. When there is a search request from a user, it reads the original HTML source at the server side. If the search word is found in the Web page, the search engine will highlight the word found. Then, it will send the reformatted Web page, after using the highlight effect, to client browser.

The core-search algorithm of the server-side approach is the same as the client-side approach. Unlike the client approach, the Character Image Search Engine is separate from the Native Input Method Engine. After the user inputs the target search text into the Input Method Engine, the Input Method Engine passes the information to the Pelutech server. They communicate with each other through their interfaces. Pelutech performs the search using the original HTML document, and then reformats the output page containing the highlighted search result. Then, the reformatted page is sent to the user, and the client browser is triggered to refresh for the newly converted page.

In addition, CISE can also accept search requests from other input method engines or client browsers. So it is not necessary for the Character Image Search Engine to be integrated with the Input Method Engine, and this reduces the download time for the user.

8.3 Experiments and Discussion

In this section, we evaluate the performance and accuracy of the Character Image Search Engine. The following experiments show how the search engine deals with different kinds of target search words. The five test cases are:

(i) Multilingual character images mixed together.

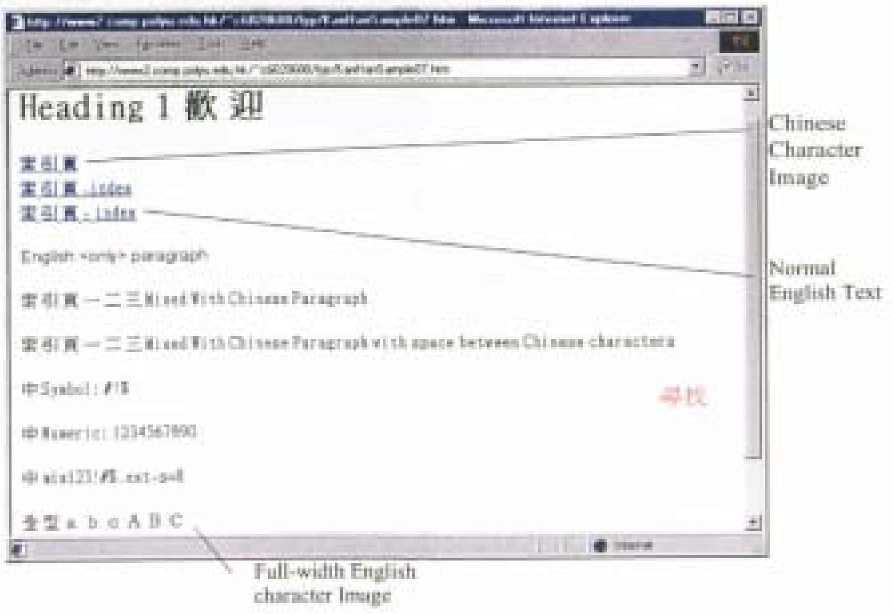
(ii) English Text with character entity references.

(iii) Full-width English character images.

(iv) Partial English word search.

(v) Search word does not exist in the Web page.

A sample test page is shown in Figure 8-5 to illustrate the test cases. This test page contains mixed character images and normal text, so that it can be used to carry out different test cases. The testing method is described briefly through Sections 8.3.1 to 8.3.5, and the experimental results are discussed in Section 8.4.


Heading 1 歡 迎

索引頁

索引頁 Index

索引頁 - Index

English <only> paragraph

索引頁一二三Mixed With Chinese Paragraph

索引頁一二三Mixed With Chinese Paragraph with space between Chinese characters

中Symbol : #!$

中Numeric: 1234567890

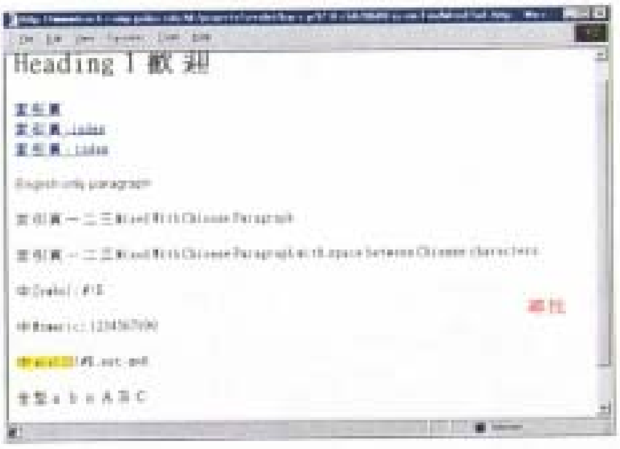
中abc123!#$.ext-a=8

全型 ａ ｂ ｃ Ａ Ｂ Ｃ

Figure 8-5: Test Web Page mixed with character image and normal text

8.3.1 Multilingual Character Images Mixed Together

In this test case, the Character Image Search Engine looks for words: "中 mix123" which is in mixed-mode: Chinese character images and English character images. The sample search result is shown in Figure 8-6 below.



Heading 1 歡迎

[illegible]

English only paragraph

[illegible] 一二三 Mixed With Chinese Paragraph

[illegible] 一二三 Mixed With Chinese Paragraph with space between Chinese characters

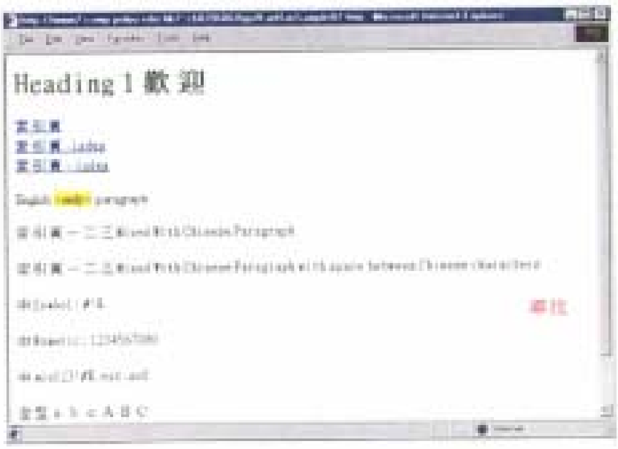
[illegible]

[illegible] a b c A B C

Figure 8-6: Search result of mixed multilingual character image.

8.3.2 English Text With Character Entity References

In this test case, the Character Image Search Engine looks for words: "<only>". It should be noted that, "<" and ">" are character entity references. In the HTML source, they are "&lt;" (<) and &gt;" (>). The sample search result is shown in Figure 8-7.



Heading 1 歡迎

[illegible]

[illegible] 一二三 Mixed With Chinese Paragraph

[illegible] 一二三 Mixed With Chinese Paragraph with space between Chinese characters

[illegible]

[illegible] a b c A B C

Figure 8-7: Search result of English character with character entity reference

8.3.3 Full-width English Character Images

In this test case, the Character Image Search Engine looks for full-width words: "Ａ Ｂ Ｃ" . The sample search result is shown in Figure 8-8.

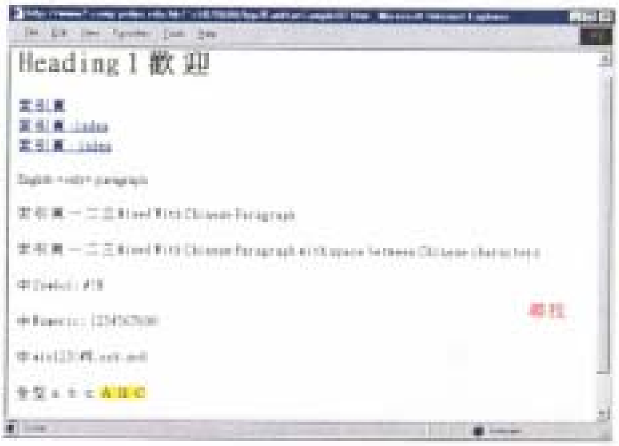

Heading 1 歡 迎

Figure 8-8: Search result of full-width English character image

8.4.4 Partial English Word

In this test case, the Character Image Search Engine looks for partial-words: "ara". The sample search is shown in Figure 8-9. It should be noted that, the word "Paragraph" and "Character" are character images and "paragraph" is normal text. As a result, with normal text only the target is highlighted but with character images the entire image is highlighted.

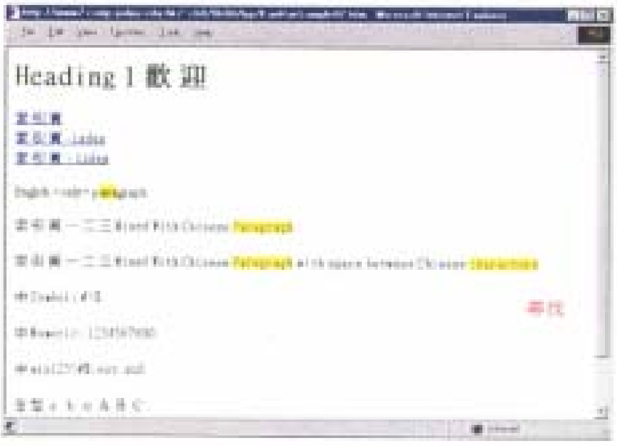
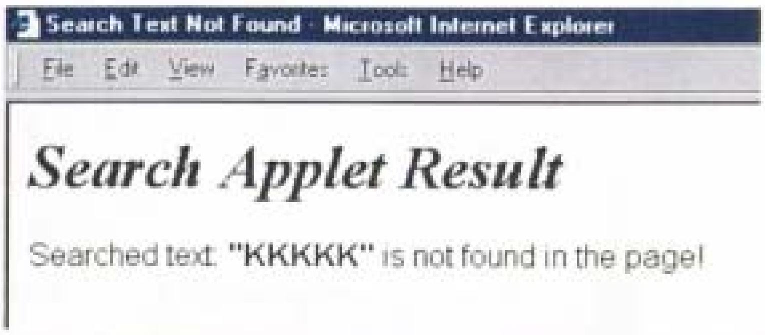

Heading 1 歡 迎

Figure 8-9: Search result of partial-word.

8.3.5 Search Word Does Not Exist in the Web Page

In this test case, the Character Image Search Engine looks for words: "KKKKK" which are not contained in the HTML document. Not found message is displayed for user in this case. The result is shown in Figure 8-10.


Search Text Not Found - Microsoft Internet Explorer
File Edit View Favorites Tools Help

Search Applet Result

Searched text: **"KKKKK"** is not found in the page!

Figure 8-10: Result of the searching word not exists in the Web page

8.3.6 Discussion on the Experimental Results

We chose ten Web pages for each of the test cases and the experimental results for the client-side approach and the server-side approach for the above five tests are shown in Table 8-2 and Table 8-3.

From the experimental results in Table 8-2 and Table 8-3, we notice that the result of the server-side approach is better than the client-side approach. The better results come from the server-side approach because it searches the original document. Therefore, the server-side approach performs a direct text-search.

We notice that there are some failures for the client-side approach in Table 8-3. The failure in search for the English text with character entity references is because that character entity reference has not been added to our character entity database. Only after we added the missing reference could we continue the test.

The failure in the full-width English character image occurs when a word has the combination of both full-width and half-width characters, such as "Ａpple" where the "Ａ" is a full-width character and remaining "pple" are half-width characters. To resolve this failure, we must convert all characters to a normalized format that is either half-width or full-width.

Test Case	Success	Fail
Multilingual characters image	10	0
English Text with character entity reference	9	1
Full-width English character image	9	1
Partial English word	10	0
Search word not exist	10	0

Table 8-2: Experimental Results for the Client-side Approach

Test Case	Success	Fail
Multilingual characters image	10	0
English Text with character entity reference	10	0
Full-width English character image	10	0
Partial English word	10	0
Search word not exist	10	0

Table 8-3: Experimental Results for the Server-side Approach

8.4 Evaluation of the Client and the Server Approaches

Both the client-side and server-side search algorithms are similar in the way that they use the Character Image Search Engine design. However, it should be noted that the client host machine is an unknown factor in the client approach. Pelutech cannot know the hardware and software configuration of the client machine before sending the re-formatted Web page. For the hardware thin client that has limited hardware resources, memory and processing power, it is not desirable to execute the Character Image Search Engine at the client side. Especially, when the reformatted Web page contains a lot of content, the search process could use up most of the system memory and CPU time because it needs to parse the HTML source and store it in memory.

Moreover, the software thin client browser may not execute the Character Image Search Engine as it is designed and implemented. Different versions of a Web browser often have different support for Java. For example, version 3.x Web browser only support Java version 1.0. Even if the Web browser supports the correct version of Java, the client side may have other Java enabled plug-ins installed that change the behavior of the working environment. Therefore, the behavior of the Character Image Search Engine could sometimes be not as expected.

The server approach is a feasible solution to reduce the uncertainties at the client side, although the server may demand more resources. For the server approach, the Character Image Search Engine runs on the Pelutech server that is under the developer's and the administrator's control. The system administrator or developer can monitor the loading of the Pelutech server and assign resources for the Character Image Search Engine.

In summary, the server approach is preferred in the thin client environment. It saves resources and time at the client side.

9. Evaluation and Discussion

In the previous chapters, we have presented a discussion on the issues that affect the performance of each of the components in Pelutech, and we have also presented some experimental results on the performance of each component. In this chapter, we present a series of experiments that are used to evaluate the overall performance and usability of Pelutech. The experiments are conducted in two parts, in which we aim to evaluate Pelutech separately for software thin-clients (part one) and for hardware thin-clients (part two).

9.1 Evaluation of Software Thin-Client Devices

To simulate a software thin-client environment, an old model of PC was set up to run the tests. The configuration of this machine is shown in Table 9-1 and the configuration of Pelutech server is listed in Table 9-2. Please note that the testing machine is not only a low-end hardware specification, but also a low-end operating system (Windows 95), which supports only English. The only additional software available on this machine is Internet Explorer 5.0. The main reason for this configuration for the testing machine is to simulate a thin-client environment with less computation power and storage capacity.

CPU:	Intel 80486 DX2 – 66MHz
RAM:	16MB RAM
Hard Disk:	420MB
Modem:	56K Modem
Operation System:	Windows 95 (English)
Internet Browser:	Internet Explorer 5

Table 9-1: Hardware and software configuration for the test platform.

CPU:	Intel Pentium III 500 MHz
RAM:	128MB RAM
Modem:	Cable Modem (maximum 1.5Mbps)
Operation System:	Windows 2000 Server

Table 9-2: Hardware and software configuration for Pelutech

The tests are designed to access different CJK Web pages to see whether our Pelutech server, which is installed on a different host, can support viewing CJK pages via the Web browser. We used the Hong Kong SAR Government's Web sites to perform these tests for Chinese and Yahoo's Japan Web site for Japanese. For systems that support CJK characters, the results should be very similar to those shown in Figure 9-1. However, since the test machine does not support the display of any CJK characters, all the CJK characters become unreadable, as shown in Figure 9-2. When we use Pelutech as a proxy server, the Chinese characters as well as the Japanese Hiragana and Katakana is correctly displayed after they are converted to character images, as shown in Figure 9-3. Note that, not only all of the CJK characters are readable, the converted Pelutech Web pages also look similar to the original pages (Please refer to Figure 9-1 and Figure 9-3).

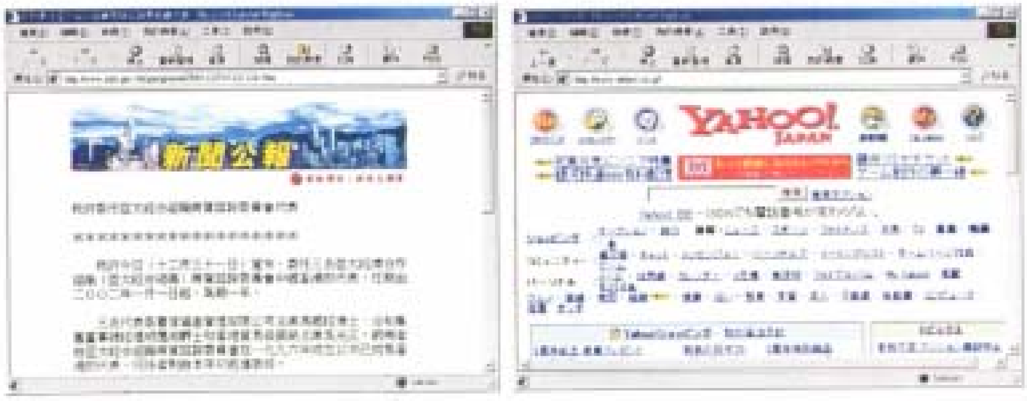


Figure 9-1: Original testing page on a CJK supported system

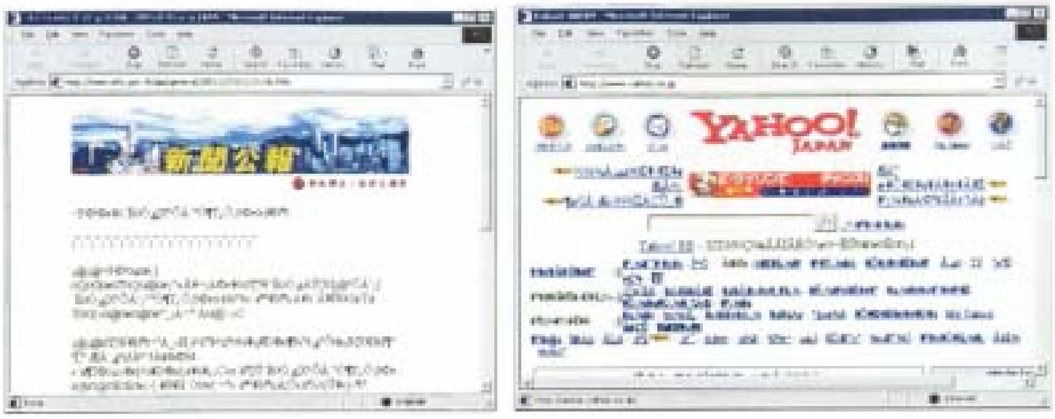


Figure 9-2: No CJK character can be displayed on the testing machine

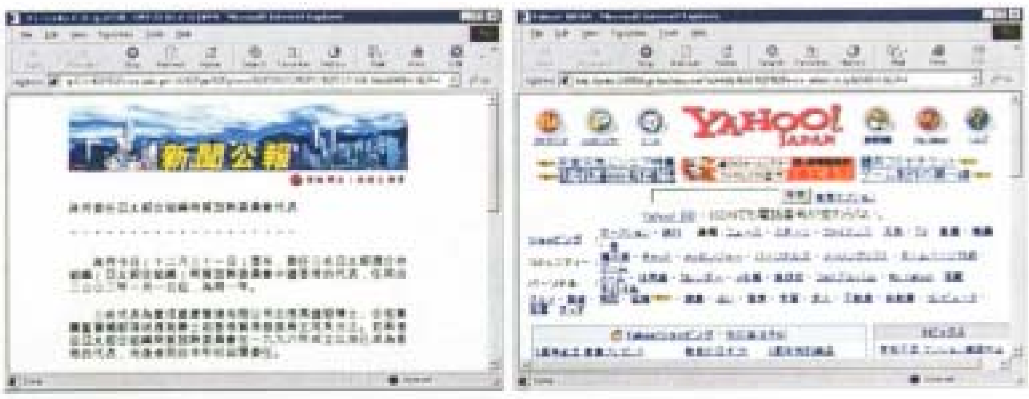


Figure 9-3: Experimental result of the testing page via Pelutech

We also performed the test on a set of 100 different Web pages using the seven performance indicators that are listed in Table 9-3. From this table, we can observe that only a single failure occurs in all the tests, and that failure is on a character image search. The main reason for that failure is that the target word is a combination of both full-width and half-width characters from English, such as"Ａpple". In this case, "Ａ" is a full-width character and "pple" are half-width characters. This is a known limitation of our current system and can be fixed by implement a converter between full-width characters and half-width characters.

Feature	Success	Fail
Original Image display	100	-
CJK character display	100	-
Hyperlink function	100	-
Client-side Installed Font Detection	100	-
Requested Web-page Encoding Detection	100	-
Searching in Page (Server Approach)	99	1
Input Method Engine	100	-

Table 9-3: Experimental results for 100 Web pages for the software thin-client

9.2 Evaluation of Hardware Thin-client Devices

To test the adequacy of Pelutech on hardware thin-clients, we conducted a set of tests using a personal digital assistant (PDA). We chose a Palm IIIc because it can support up to 256 colors. To enable wireless Internet browsing, a Nokia 8250 was used as the external modem. By using its infrared communication capabilities, the Palm IIIc can use the internal modem in the mobile phone to gain access to the Internet. The hardware configuration of this testing platform is shown in Table 9-4.

Model:	Palm IIIc
Memory:	8MB
Operation System:	Palm OS v3.5 (English)
Internet Browser:	PalmScape 3.03E
Modem:	Nokia 8250 internal modem

Table 9-4: Hardware and software configuration for the test PDA platform

When we use the Palm PDA to view the Hong Kong Government's web site, the Chinese characters cannot be displayed correctly without Pelutech. As shown in Figure 9-4, all the Chinese characters become unreadable symbols. When we use Pelutech as the proxy server to view the Web site again, the Chinese characters can be correctly displayed, as shown in Figure 9-5. Please note that in the Web page on the left hand sides, the second line of the frame below the headline picture is a bitmap that is not being converted.

We use the same set of Web pages to test the other functions. The summary of the results is given in Table 9-5.

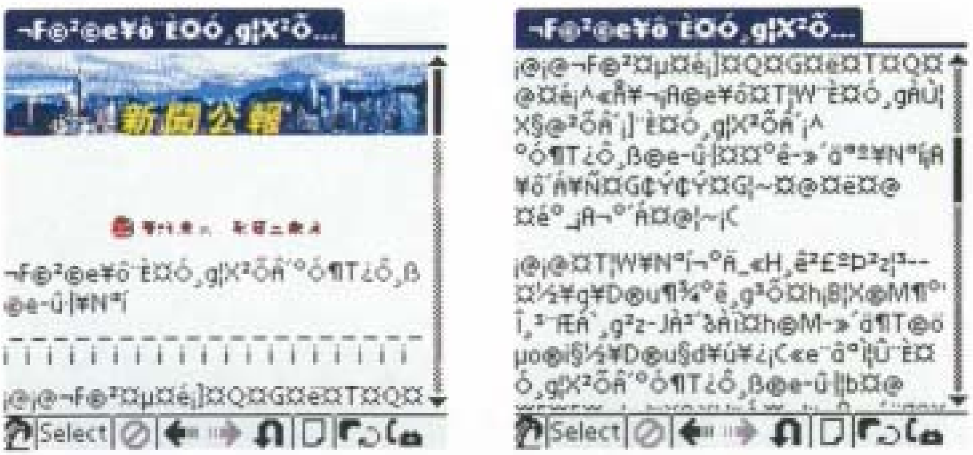
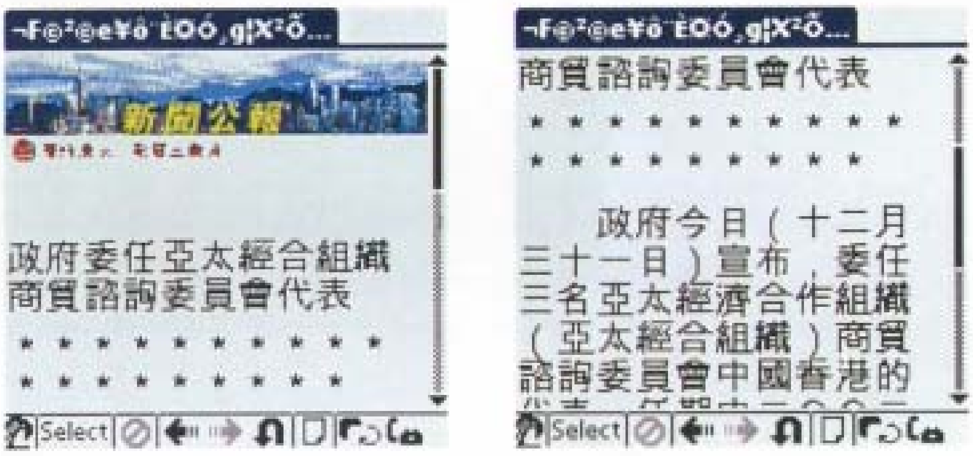


Figure 9-4: The CJK character cannot be displayed on the PDA



政府委任亞太經合組織商貿諮詢委員會代表

＊ ＊ ＊ ＊ ＊ ＊ ＊ ＊ ＊ ＊

＊ ＊ ＊ ＊ ＊ ＊ ＊ ＊ ＊

商貿諮詢委員會代表

政府今日（十二月三十一日）宣布，委任三名亞太經濟合作組織（亞太經合組織）商貿諮詢委員會中國香港的

Figure 9-5: The display of the testing page through Pelutech

Feature	Success	Fail
Original Image display	100	-
CJK character display	71	29
Hyperlink function	100	-
Client-side Installed Font Detection	-	100
Requesting Web-page Encoding Detection	100	-
Searching in Page	-	100
Input Method Engine	-	100

Table 9-5: Experimental Result for 100 Web pages for hardware thin-client

We found that Pelutech failed completely in three functional tests using the PDA: Client-side Installed Font Detection, Searching in Page, and Input Method Engine. The failure of these three functions is due to the lack of support for HTML in PDA environments. All of these three functions require the support of the HTML 4.0 specification, while all existing Internet browsers for the Palm OS only support versions up to HTML 3.2, therefore, these functions cannot be executed properly. However, with future support for HTML 4.0 or above in a PDA browser, we do not foresee any problem in supporting these functions because they are software dependent, not hardware dependent.

9.3 Analysis of the Response Time of Pelutech

Pelutech acts as a proxy between a client and a Web server, as shown in Figure 9-6. Pelutech causes an extra relay requesting a Web page to client from Web server when compared to a direct connection from the client to a server. In this section, we analyze the delay caused by Pelutech (see Figure 9-7) when a client connects to a Web server though Pelutech.

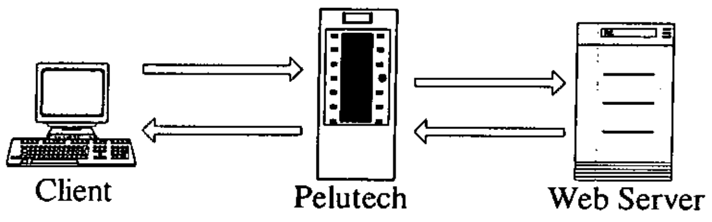


Figure 9-6: Request and response interaction through Pelutech

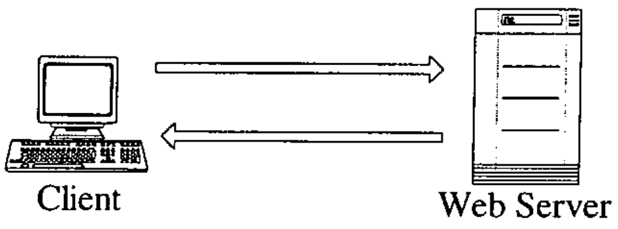


Figure 9-7: Normal request and response interaction without Pelutech

Before evaluate the actual performance, we first calculate the response time based on the analytical calculation. In this calculation, we first assume that the latency caused by the slow-start effect and connection tear down, and the *Round Trip Time* (also called TCP connection time) as well as the server processing time is constant. The description of each variable are listed in Table 9-6.

Term	Description
B	Average transmission speed of the network (bits per second)
C	Delay cause by the slow-start effect and the Round Trip Time as well as the server processing time for Normal Browsing
C'	Delay cause by the slow-start effect and the Round Trip Time as well as the server processing time through Pelutech
N	Number of CJK character in the web document
L	Size of original web page
α_i	Size of the character image
β_i	Size of the additional code per character

Table 9-6: Description of each variable

For a Web site comprises one main HTML page, which contains N CJK characters, stored in L bits. We assume that the average transmission speed of the network is B bps, Therefore, the transmission time, T_{Normal}, to retrieve the original Web page is:

$$T_{Normal} = \frac{L}{B} + C$$

In other word, the response time for normal web browsing is proportional to the page length and invert proportional to the bandwidth plus a contact factor related to the server processing time and delay. We label it as Baseline Model.

When browsing through Pelutech with conversion, an extra delay is caused because the total distance is composed the propagation delay between client and Pelutech and the propagation delay between Pelutech and web Server. In addition, since the HTML code is different after conversion, with converted character images, the page length of the converted HTML will be extended to $\sum_{i=1}^{N}(\alpha_i + \beta_i)$ where β_i is the size of the additional modified HTML code that is required to support each CJK character as an image and α_i is the size of the converted character image. Furthermore, when a user connects to a Web server, his connection is via Pelutech. There is an additional request and response time during the communication and processing between Pelutech and the Web server. Therefore, the response time, $T_{Pelutech\_conversion}$, for a user connection to retrieve the converted Web page from a Web server via Pelutech is equal to the summation of two delays. Thus,

$$T_{Pelutech\_conversion} = \left(\frac{L}{B}\right) + \left(\frac{L + \sum_{i=1}^{N}(\alpha_i + \beta_i)}{B}\right) + C'$$

$$= \frac{2L + \sum_{i=1}^{N}(\alpha_i + \beta_i)}{B} + C'$$

We found that the response time is directly proportional to the sum of the original page length L and the modified page length $\sum_{i=1}^{N}(\alpha_i + \beta_i)$. Assuming every character image has the same size, $\alpha_i + \beta_i$ can be substituted by ϕ. The response time becomes:

$$T_{Pelutech\_conversion} = \frac{2L + N\phi}{B} + C'$$

If the client contains the required font for the Web page so that no conversion is needed ($\phi = 0$), the response time is reduce to

$$T_{Pelutech\_no\_conversion} = \frac{2L}{B} + C'$$

If we assume that C' is twice as long as C, then

$$\begin{aligned} T_{Pelutech\_no\_conversion} &= \frac{2L}{B} + 2C \\ &= 2\left(\frac{L}{B} + C\right) \\ &= 2T_{Normal} \end{aligned}$$

This is the best case, Pelutech will introduce twice the delay as the normal browsing.

The time difference, $D_{Conversion}$, between retrieving a Web page directly from a Web server and retrieving a converted Web page through Pelutech is:

$$\begin{aligned} D_{Conversion} &= T_{Pelutech\_conversion} - T_{Normal} \\ &= \left(\frac{2L + N\phi}{B} + C'\right) - \left(\frac{L}{B} + C\right) \\ &= \left(\frac{2L + N\phi}{B} + 2C\right) - \left(\frac{L}{B} + C\right) \\ &= \frac{L + N\phi}{B} + C \end{aligned}$$

If the original page is in English or the client contains CJK fonts, there is no need to perform the conversion. The Time difference, $D_{No\_conversion}$, is then:

$$\begin{aligned} D_{Conversion} &= T_{Pelutech\_no\_conversion} - T_{Normal} \\ &= \left(\frac{2L}{B} + C'\right) - \left(\frac{L}{B} + C\right) \\ &= \left(\frac{2L}{B} + 2C\right) - \left(\frac{L}{B} + C\right) \\ &= \frac{L}{B} + C \\ &= T_{Normal} \end{aligned}$$

To calculate the delay that affects the user when they use Pelutech, we calculate the response time for different transmission speeds (bandwidths): 9.6 Kbps (mobile internal modem speed), 56 Kbps (computer modem speed), 128 Kbps (GPRS — general packet radio service) and 1.5 Mbps (broadband speed) for Web page with different number of CJK characters. General speaking, the normal number of CJK characters in Web page can be classified into follow catalogues: (1) a notice window contains about 50 CJK characters; (2) a normal Web page contain about 100 to 200 CJK characters; (3) a longer Web page such as story or comment often contains near 500 CJK characters; (4) and the longest Web page contains as most as 1,000 CJK characters.

We assume that the normal page length of a Web page is 1K bytes, and each CJK character is 2 bytes long. Therefore, we use five different scenarios: (i) 50 CJK characters *(N)* inside a Web page, and the page size (*L*) is assumed to be 1.1 Kbytes; (ii) 100 CJK characters inside a Web Page, and the page size (*L*) is assumed to be 1.2 Kbytes; (iii) 200 CJK characters inside a Web Page, and the page size (*L*) is assumed to be 1.4 Kbytes; (iv) 500 CJK characters inside a Web Page, and the page size (*L*) is assumed to be 2 Kbytes; (v) 1,000 CJK characters inside a Web Page, and the page size (*L*) is assumed to be 3 Kbytes. In addition, the average size of a converted image (α) is assumed to be 1 Kbytes, because the actual size is between 760 – 1280 bytes. The average size of the additional code (β) is assumed to be 20 bytes (the actual code size is between 17 – 22 bytes). The response times (in seconds) for direct connection and proxy connection via Pelutech are shown in Figure 9-8 and Figure 9-9, respectively. Figure 9-10 shows the response via Pelutech when no character conversion is required. The analytical data is for Figure 9-8 to Figure 9-10 is listed in Table A-1, Table A-2 and Table A-3 respectively in Appendix I.

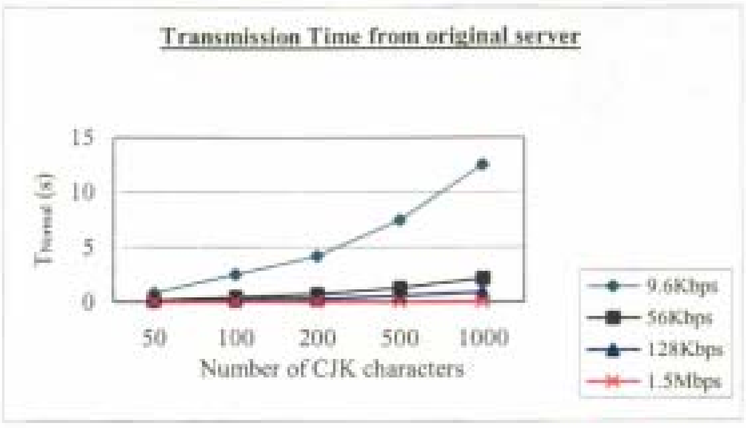


Figure 9-8: Transmission time for retrieving Web pages from a Web server at different bandwidths

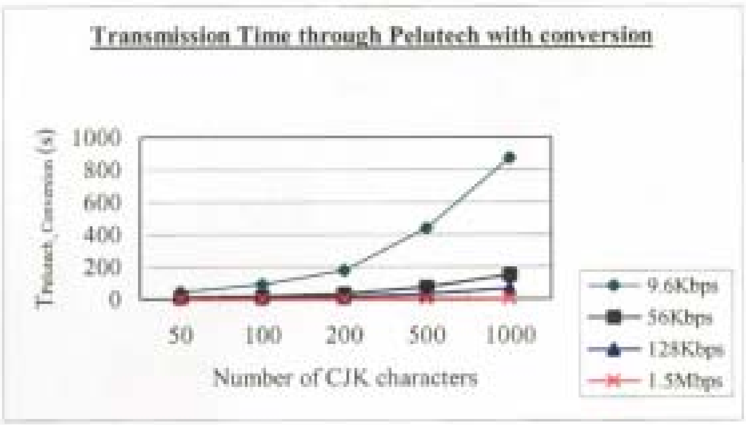


Figure 9-9: Transmission time for retrieving Web page through Pelutech (with conversion) at different bandwidths

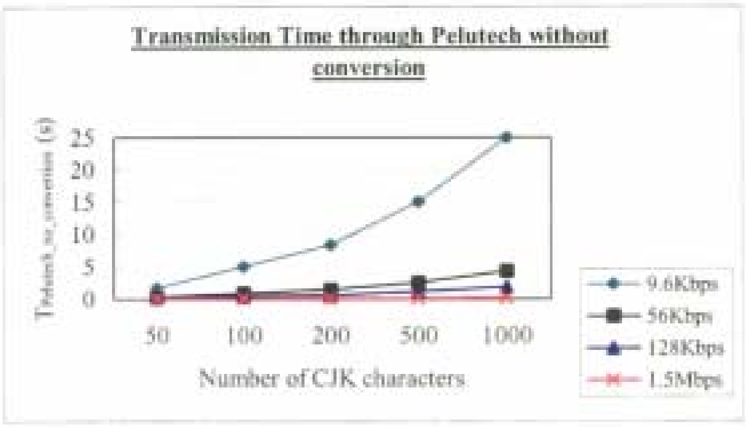


Figure 9-10: Transmission time for retrieving Web page through Pelutech (without conversion) at different bandwidths

To evaluate the real performance of Pelutech, we perform actual response time tests to examine the validity of the analytical model presented above. We use the PDA for the experiment at 9.6 Kbps, since the current internal modem technology can only support up to 9.6 Kbps in a mobile phone, for the hardware thin-client environment. We use a computer modem and a broadband connection to test the speed at 56 Kbps and at 1.5 Mbps for software thin-client environments. Figure 9-11, Figure 9-12 and Figure 9-13 show the experimental results for retrieving a Web page directly from a Web server, retrieving a page via Pelutech with conversion, and retrieving a page via Pelutech without conversion, respectively and the experimental data is shown in Table A-3, Table A-4 and Table A-5 in Appendix respectively. Note that, our test Web page is generated using a special program so that each character within the same page is unique, and Pelutech has to transfer an image for every character.

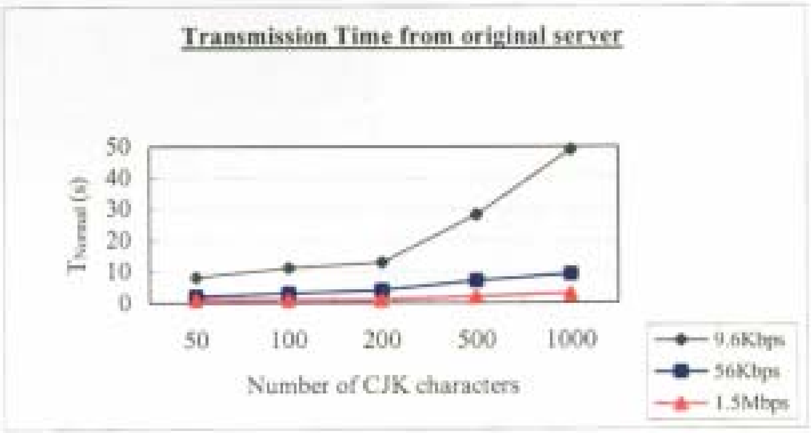


Figure 9-11: Transmission Time for retrieving a Web page directly from a Web server at different bandwidths

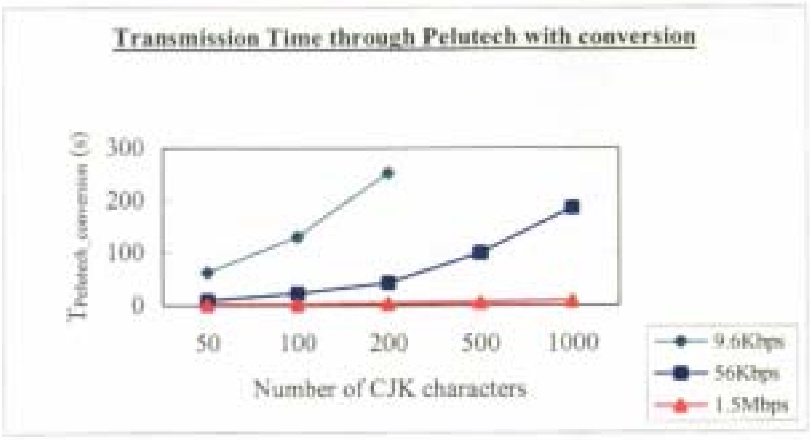


Figure 9-12: Transmission Time for retrieving Web page through Pelutech (with conversion) at different bandwidths

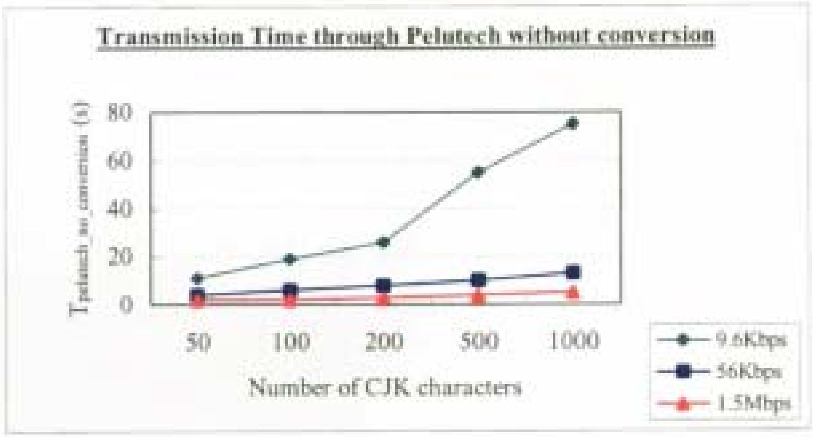


Figure 9-13: Transmission Time for retrieving a Web page through Pelutech (without conversion) at different bandwidths

From all of the graphs above, we can see the performance of Pelutech is near linear at the fixed page size up to 50KB. This means that the transmission time increases linearly with the number of characters transmitted.

In general, it is possible that when the page sizes become so large, the response time will deteriorate much faster. However, consider that most of Web page size for thin-clients is always smaller than the normal Web page because the Web page is normal reconstructed at a much smaller file, say less than 10KB. The near linear performance is reasonable.

In addition, we find that a "time out" error occurs when browsing the Web page that contains 1,000 CJK characters. After examining the execution of this test in detail, we found that the error was due to the Palm Web browser server. For Web browsing on a Palm OS device, the current technology uses a transparent proxy to convert the Web page content to a Palm supported format, such as converting images to the "pdb" format. Therefore, when browsing the Web page containing 1,000 CJK characters, Pelutech first converts the CJK characters to character images. The Palm browser proxy then retrieves those 1,000 CJK character images and converts them to the "pdb" format. This is a time consuming activity for the Palm browser server, and a "Time out error" occurs when the Palm browser cannot retrieve the data from its own proxy within a certain time limit. The normal time out error occurs after waiting 300 seconds.

To solve this problem, we can separate the Web page into several smaller and more manageable sets of data. Since the screen size of the current PDA can show only a limited part of the Web page (without scrolling), we can deliver the Web page content to a PDA in pieces rather than send the entire page [62]. For example, a particular Web page requires 5 screens of content to show the whole page. We can deliver the first 2 screens of content to the browser first, and as the user starts to scroll down the screen the server delivers further screens of content (until the page is complete). By using this method, the waiting time for a user can be decreased significantly, and it also reduces the overhead of using a proxy server for the Palm browser.

As shown by the experimental results above, the response time for a user browsing via Pelutech is more than 7 times the direct Web server response. However, we must also take into account that the experimental results involve pages that contain only unique CJK characters. In other words, 200 conversions are needed for a page containing 200 CJK characters. Although in practical cases, for a Web page containing 200 CJK characters, there might only be 100 to 150 unique characters. For larger pages containing over 1000 CJK characters, we usually find that less than 300 characters are unique. Consider the text example shown in Figure 9-14, even the "一" and "十" characters each occur 4 times in this short essay containing a total of 180 Chinese characters. Other characters are also duplicated several times in this short essay.

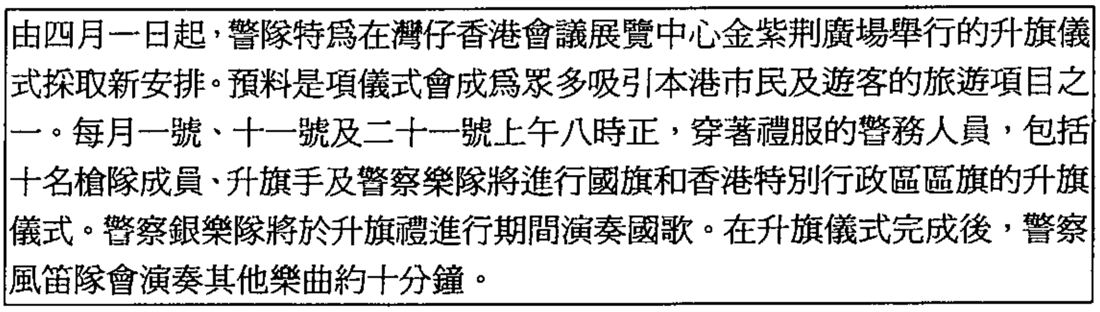

由四月一日起，警隊特爲在灣仔香港會議展覽中心金紫荊廣場舉行的升旗儀式採取新安排。預料是項儀式會成爲眾多吸引本港市民及遊客的旅遊項目之一。每月一號、十一號及二十一號上午八時正，穿著禮服的警務人員，包括十名槍隊成員、升旗手及警察樂隊將進行國旗和香港特別行政區區旗的升旗儀式。警察銀樂隊將於升旗禮進行期間演奏國歌。在升旗儀式完成後，警察風笛隊會演奏其他樂曲約十分鐘。

Figure 9-14: Partial Web page extracted from the Hong Kong SAR Government's Web site

Since the occurrences of the characters in different Web pages are obviously different, it is difficult to calculate the average percentage of unique characters. However, we can conclude that the proportion of characters that are unique in a page will generally decrease as the page size increases. Therefore, the practical response times when using Pelutech will be slightly less than the experimental results seem to suggest.

To evaluate the actual situation of browsing through Pelutech, we perform actual response time experiments again by visiting the 20 web pages for each test cases: 50, 100, 200, 500 and 1,000 CJK characters above. Then each web page is visited by 10 times and the average time was taken for calculation. Since it is very difficult to obtain a web page that contains exactly 50, 100, 200, 500 and 1,000 characters, we use the web pages that contains ± 10% deviation. The number of characters for each test case is listed in the Table 9-11 below.

Test Case	Number of characters in the tested Web page
50 CJK characters	45 – 55
100 CJK characters	90 - 110
200 CJK characters	180 - 220
500 CJK characters	450 - 550
1,000 CJK characters	900 - 1100

Table 9-7: Number of Characters in the Tested Web Page for each Test Case

By repeat the above experiments using these test cases, we have obtained the actual performance data shown in Figure 9-15 to Figure 9-17 and the experimental data are listed in Table A-7 to Table A-9 in Appendix.

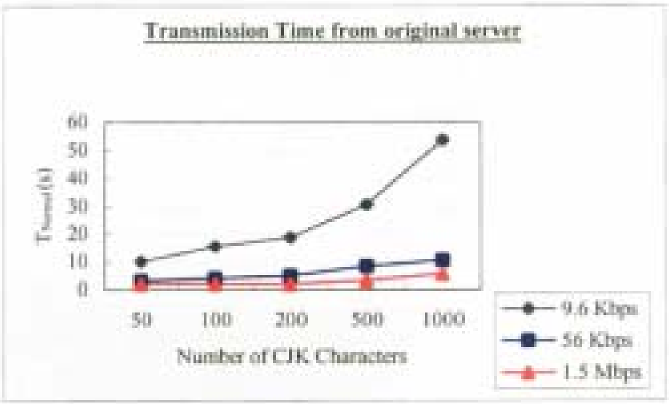


Figure 9-15: Transmission Time for retrieving Web page from original server

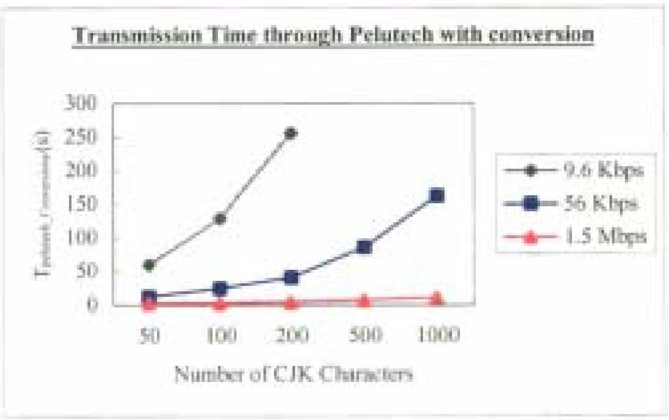


Figure 9-16: Transmission Time for retrieving Web page through Pelutech with conversion

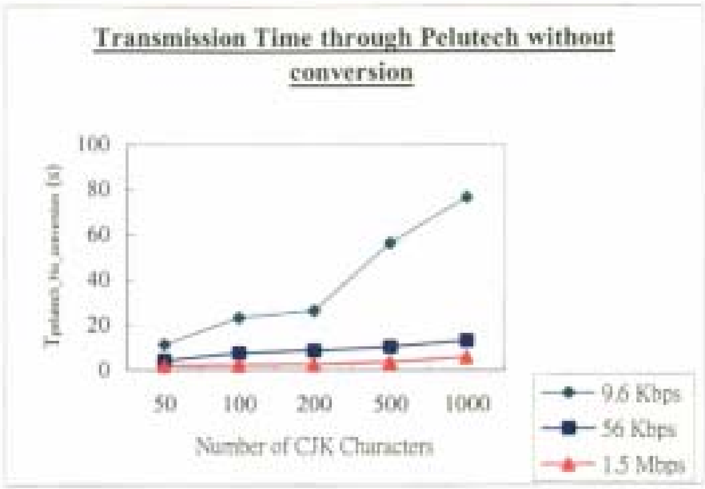


Figure 9-17: Transmission Time for retrieving Web page through Pelutech without conversion

From the graph above, we found that some results are a little bit lower than the performance in Figure 9-11 to 9-13. The reason is that the testing page in the previous experiment are generated by a program with the same HTML code embedded, while the Web pages are not the same in this set and the HTML code embedded is different.

From the observation in the above result, we also found out that the downloading time for 500 and 1,000 CJK characters are less than those in the previous experiments. This is due to the duplication of the character in the essay. The fact when the length of a Web page increase, the number of duplicated characters increase too. This can greatly decrease the downloading time for the converted page.

On the other hand, we have compared the difference of the delay in response time between analytical model and experiments and the diagram is shown in Figure 9-18.

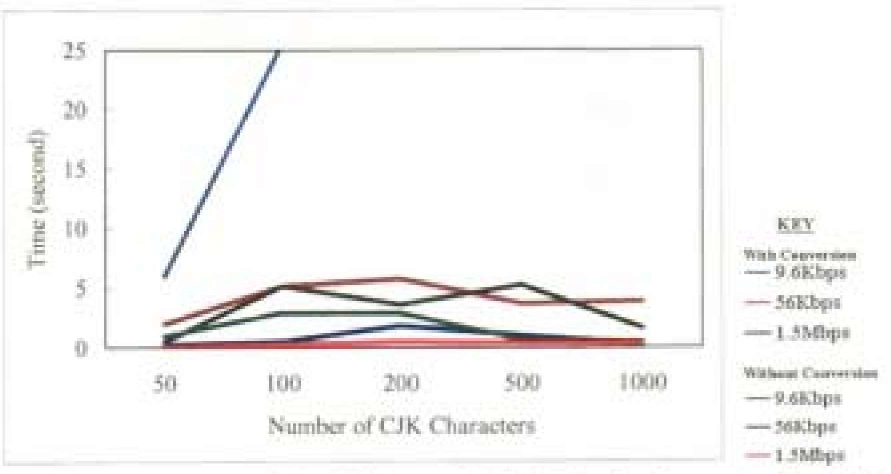


Figure 9-18: Comparison of the Different of the Delay in Response time between Analytical Model and Experimental Results

When the client browses through Pelutech without conversion, we found that the experimental result is very close to the analytical model. For the case that the client browses through Pelutech with conversion, the difference between the analytical model and the experimental result is nearly constant except the experimental result by the 9.6Kbps modem. This is because we using the current browser in PDA require an additional proxy to handle the HTML document in current stage. In conclusion, the actual result is acceptable because the transmission time is near linear.

9.4 Server Utilization and Scalability

Beside the evaluation of the performance in the delay of response time, we also need to consider server utilization and scalability of Pelutech. Server utilization indicates how busy the server CPU. When it is close to 100% utilized, jobs will have to be queued up, and thus it directly affects the response time to client request. Scalability refers to the system's ability to expand its support on an increasing number of concurrent users without degradation of performance. Scalability is best accomplished when Pelutech is hosted on a cluster of servers. When a request comes in, that request is routed to the least busy processor.

To evaluate server utilization, we can use the stochastic model in queuing theory to see when the system will be saturated. By *Saturation* we mean when the number of requests to the server reaches certain level at regular time intervals, the CPU utilization will reach a saturation point where additional requests may not be handled in first in first out fashion anymore. In other words, the system will not be able to respond to their requests. At this point, system must spawn additional servers to entertain these requests.

In order to have a good estimate of CPU time for each web request, we send a number of requests to Pelutech simultaneously. In this experiment, we turn off the thread inside Pelutech and send out 50 conversion requests from client to Pelutech, and count the time that requested to finish all conversion sequentially. The number of characters in each Web page is 1,000 characters. We found that the CPU use 13 seconds to finish 50 incoming conversion tasks. This means Pelutech can finish each conversion in 26ms. In other word, the service rate of Pelutech is 1/0.26 (3.85) jobs per second. This is the best-case scenario because the server is dedicated to handle one job only. To consider scalability issue, the CPU utilization is can be set to a rate of no more than 80% for handling conversions. Other overhead will also consume CPU power such as thread management, fragmentation, switching, or other operating system related overhead. Since the server utilization p for a single processor is $p = \frac{\lambda}{\mu}$, where λ denotes the arrival rate and μ denotes the service rate. For 80% utilization,

$$\frac{\lambda}{\frac{1}{0.26}} \leq 80\%$$

$$\lambda \leq \frac{40}{13}$$

Therefore, the arrival rate should be less than 40/13 (3.08) jobs per second in order to have a maximum of 80% server utilization. In other word, if there are more than 185 incoming requests per minute, more powerful processors is recommended or spawning to additional servers must be started.

Another factor that would affect scalability issue is the response time to client requests because most Internet browser automatic time-out a request at 300 seconds (or 5 minutes) by default. In other word, a client browser must download all required file within 300 seconds in order to display the whole page for user correctly. Otherwise, resubmission is needed and previous submissions are considered lost even if they are still in the queue at the server side.

In order to have a good estimate of response time for each web request versus the number of concurrent users (or requests), we send a number of requests to Pelutech simultaneously. In this experiment, the number of threads in Pelutech is set to 10 and the number of CJK characters in the requested web page is 1,000.The number of threads is set to 10 here because Pelutech must be a multi-thread server in production environment. In addition, the experimental result will not be affect by the number of threads in a single CPU environment. The experimental results for the average response time needed for user against the number of concurrent users per second for Pelutech with and without conversion are shown in Figure 9-19 and Figure 9-20 and the experimental data is listed in Table 9-8 and Table 9-9.

Number of concurrent users (per second)	1	2	3	4	5	6	7	8	9	10
Average Times needed (s)	10	11	13	15	32	103	290	Error	Error	Error

Table 9-8: Response time against No. of concurrent users with conversion

Number of concurrent users (per second)	1	2	3	4	5	6	7	8	9	10
Average Times needed (s)	5	6	7	13	19	25	45	112	263	Error

Table 9-9: Response time against No. of concurrent users without conversion

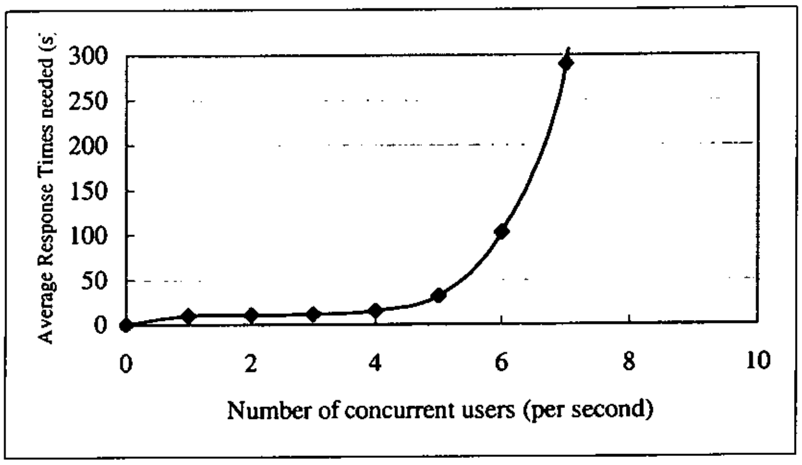


Figure 9-19: Response time against the no. of concurrent user with conversion

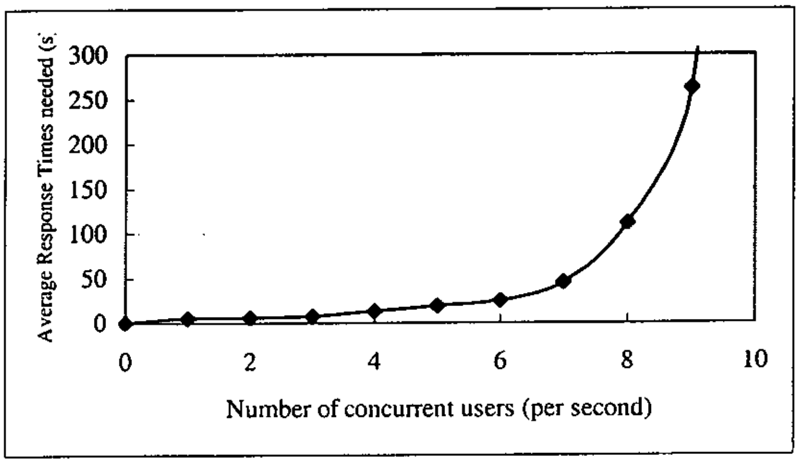


Figure 9-20: Response time against no. of concurrent users without conversion

Note that the rate of change in the average response time against the number of concurrent users is changed from a near constant to a step up function f (x) when there are more than 4 concurrent users per second (240 concurrent users per minutes) in both figures. Besides, as shown in Table 9-8, we find that the client browser get a time out error when there are over 7 concurrent users per second (420 concurrent users per minute). Moreover, as shown in Table 9-9, the time out error also occurs when there are over 9 concurrent users per second (560 concurrent users per minute) without conversion. This indicates the system will become saturated at this point. Additional CPU is recommended to install for handling extra process.

We can increase the number of threads in a multi-CPU environment to improve the system throughput. However, when the number of threads is increased, larger memory space is needed for the creation of server process. In our observation, we found that each additional thread required 5~10MB memory resource. Therefore, with memory space of 512MB, the server can in principle support up to 50 threads since the Operating System also needs about 150MB. In reality, however, considering the saturation point of no more than 10 current requests as resulted from the above experiment, there is no need to set the number of threads over 10. From the above experiments we can see that system can give very reasonable response time when concurrent requests are around 4 per second (about 240 concurrent uses per minutes). It would start to deteriorate until the number of concurrent requests reaches 7 per second (about 400 concurrent uses per minutes). Hypothetically speaking, if one expects 1,000 connections per minute, a four processors server cluster is suggested for life production because the maximum capacity is up to 1,600 concurrent users.

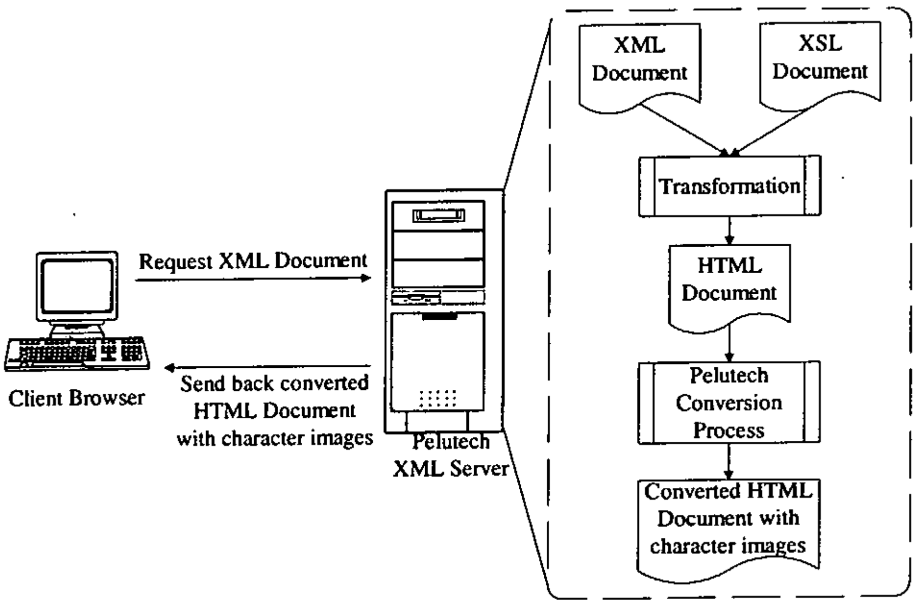
10. Conclusion

In this thesis, we have presented a novel server-based technique to support CJK character display using character images, and implemented this technique in the Pelutech system. This is a comprehensive Web-based thin-client technology that does not need to retrofit the existing Web pages. It provides not only an efficient way to display CJK characters without installing fonts, but also the ability to detect font sets on client machines and the codeset for the requested Web pages. In addition, the comprehensive HTML parsing and handling algorithm provides the unique ability to handle complex form controls and the provision of searching interaction during Web browsing even when characters are converted to images. In addition, it has virtually unlimited character-set support since all character images are generated on the server side. Furthermore, the provision of an Internet-based input method engine allow users to input text independent of the language environment of their Web access devices. Besides, the portable design of the input method engine provides the extension for other applications. On the other hand, it is compatible with Windows, Macintosh, UNIX, set-top boxes and other Web appliances. The end-users can potentially save the cost of purchasing the client-side localize software. Since this technology provides a frustration-free experience in accessing CJK Web sites and achieves significant cost reduction for manufacturers of localized Web appliances, it is the best solution for thin-client devices which only have limited hardware and software support.

There are still some limitations in using the current version of Pelutech. Although Pelutech provides almost full support in the software thin-client environment, there are some constraints in the support for hardware thin-client devices. One limitation is due to the screen size of the current thin-client devices. Current PDA can only support a resolution up to 320 x 240 pixels. However, Web pages are predominantly designed for desktop computers using resolutions of 800 x 600 pixels or higher [62]. The PDA browser downloads the whole Web page to the device, but the user can only view a portion at any time (roughly, this might be 10%). Therefore, there is a need to design an additional Pelutech component to automatically break down web pages into smaller pages for PDA viewing. It is also said that new PDA with higher resolution screens will be launched within a year. It is also helpful to extend Pelutech to support different XML because XML might replace HTML in the future. We do see the advantage of XML because it is syntactically more restrictive. Web pages written in XML are better formatted with less arbitrary written styles. In order to support XML, we can apply the current technique to XML for the XSL Transformation (to HTML document). There are two methodologies for this extension and the framework design for the extension of Pelutech to support XML are shown in Figure 10-1 and Figure 10-2.

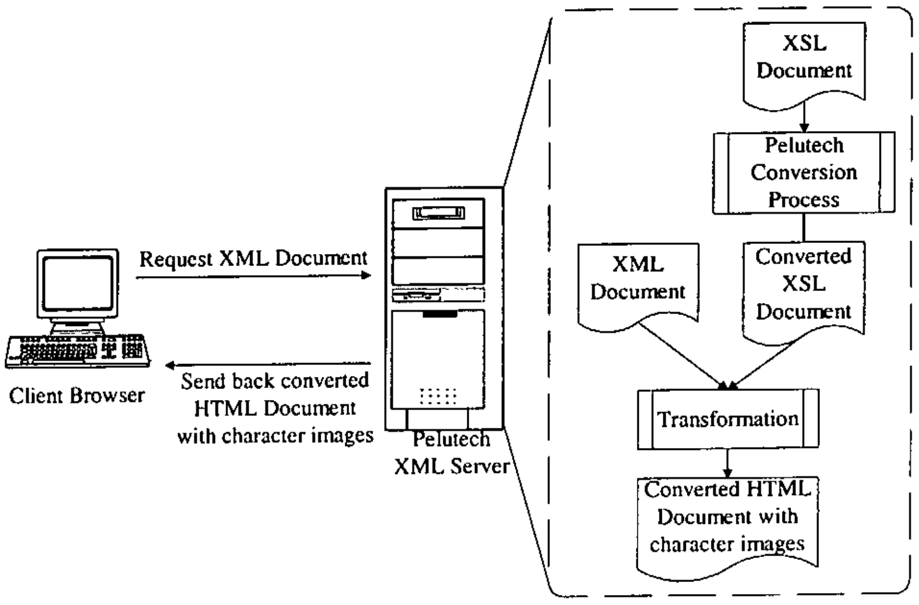
In Method 1, Pelutech can be migrated to support XML by adding a conversion process to handle the output of the HTML document after the XML Transformation so that users can read the XML page without the installation of local font. This method can be easily implemented since our Pelutech is ready to support HTML document.

In Method 2, the XSL document is converted instead of the output HTML document. Although this method sound much more complicated than Method 1, this allows us to change the user interface because we can directly handle the style sheet (XSL document).



Request XML Document
Send back converted HTML Document with character images
Client Browser
Pelutech XML Server
XML Document
XSL Document
Transformation
HTML Document
Pelutech Conversion Process
Converted HTML Document with character images

Figure 10-1: Extension of Pelutech to support XML (Method 1)



Request XML Document
Send back converted HTML Document with character images
Client Browser
Pelutech XML Server
XSL Document
Pelutech Conversion Process
XML Document
Converted XSL Document
Transformation
Converted HTML Document with character images

Figure 10-2: Extension of Pelutech to support XML (Method 2)

In conclusion, Pelutech provides a frustration-free experience in accessing CJK Web sites, and it achieves significant cost reductions for manufacturers of localized Web appliances. It is also provides the most comprehensive solution for thin-client devices which only have limited hardware or software support.

Reference

1. Xian-Ping Li, Wei Li, Chi-To Lam. "Statistics in the Study of Chinese Characters", In Proceedings of the 1997 Internet Conference on Computer Processing of Oriental Languages (ICCPOL'97), Hong Kong, 2-4 April 1997, pp.510-513 (1997)

2. Sun Jianhua, Shen Vicent Y. "Browsing and Creating Chinese Web Pages", In Proceeding of The Second Asia Pacific Web Conference (APWEB'99), Hong Kong, 28-30 Sep 1999, pp. 285 – 288 (1999)

3. Joseph T. Sinclair. Thin Clients: Clearly Explained. Academic Press (2000)

4. Barbara F. Grimes and Joseph E. Grimes, Ethnologue, Volume 1: Languages of the World, Fourteenth Edition, 2000. SIL International, Dallas, USA (2000)

5. United Nations Population Division, Department of Economic and Social Affairs World Population Prospects: The 2000 Revision - Annex Tables of the Highlights. USA (2000)

6. David K. Jordan, The Chinese Language(s), http://weber.ucsd.edu/~dkjordan/chin/hbchilang-u.html, 14 July 2001, USA (2001)

7. Edberg Peter. "Survey of Character Encodings", In Pre-Conference Tutorial of the Thirteenth International Unicode Conference, San Jose, California, 8-11 Sep 1998, TA4 pp. 1-23 (1998)

8. Fraase, Michael, The Windows Internet Tour Guide: Cruising the Internet the Easy Way. Ventana Press (1995)

9. Feldman Boris, Microsoft Internet Explorer 5 Web Programming Unleashed. Sams Publishing, Indianapolis, USA (2000)

10. Price-Wilkin John. "Using the World Wide Web to Deliver Complex Electronic Documents: Implications for Libraries". The Public-Access Computer Systems Review 5, no. 3, 1994, pp. 5-21, USA (1994)

11. TrueDoc Inc, http://www.truedoc.com/

12. Microsoft Corporation, Font embedding for the Web, http://microsoft.com/OpenType/web/embedding/, 26 Feb 2001, USA (2001)

13. Ken Lunde. CJKV Information Processing. O'Reilly. (1998)

14. Ralf Steinmetz, Klara Nahrstedt. Multimedia Computing, Communications and Applications. Prentice-Hall (1995)

15. The Internet Society, Hypertext Transfer Protocol – HTTP/1.1, June 1999, USA (1999)

16. Welch Jim. "Platform Independent Font Support", In Proceeding of the Eighth International Unicode/ISO 10646 Conference, Hong Kong, 18-19 Apr, 1996, A3 pp. 1-21 (1996)

17. Bitstream Inc, "What Are Dynamic Fonts?", http://www.truedoc.com/webpages/getstart/get_start1.htm

18. Luke Duncan and Sean Michaels, Official Netscape Technologies Developer's Guide. Ventana Communications Group, USA (1997)

19. Microsoft Corporation, Font Embedding on the World Wide Web, http://www.w3.org/Printing/pennock.html, April 19, 1996, USA (1996)

20. Siu Chi Hsu, Kin Hong Lee, Chin Lu, Man Fai Wong, and Wing Shing Wong, "The HANZIX Open System", Communications of COLIPS, An International Journal of The Chinese & Oriental Languages Information Processing Society, Vol. 5, Number 1, Dec. 1995, pp. 29 – 36 (1995)

21. Shodouka, http://web.lfw.org/shodouka/

22. MINSE PolyMediator, http://web.lfw.org/math/

23. Web Access Gateway, http://www.cus.cam.ac.uk/~ssb22/access.html

24. Silas S Brown and Peter Robinson, "A World Wide Web Mediator for Users with Low Vision", In Proceeding of the Workshops on the Conference on Human Factors in computing Systems, Washington, USA, 31 March – 5 April 2001 (2001)

25. Asmus Freytag, "Character Set Guessing". In Proceeding of the Thirteenth International Unicode Conference, San Jose, California, 8-11 Sep, 1998, B7 pp. 1-23 (1998)

26. Alan J. Dix, Janet E. Finlay, Gregory D. Abowd, Russell Beale, Human-Computer Interaction, Second Edition. Prentice Hall, New York (1998)

27. Martin J. Durst, Gavin Thomas Nicol, Francois Yergeau, "Weaving the Multilingual Web: Standards and their Implementation". In Proceeding of the Sixteenth International Unicode Conference, Amesterdam, The Netherlands, 27-30 March 2000, TA3 pp.1-48 (2000)

28. Cathyann Swindlehurst, "Globalization, Internationalization and Localization: An Introduction". In Proceeding of the Sixteenth International Unicode Conference, Amesterdam, The Netherlands, 27-30 March 2000, TC1 pp.1-16 (2000)

29. Balachander Krishnamurthy, Jennifer Rexford, Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement. Addison Wesley Longman, Inc (2001)

30. Yean Fee Ho, "Creating a Multilingual Unicode WWW Homepage". In Proceeding of the Eighth International Unicode/ISO 10646 Conference, Hong Kong, 18-19 April 1996, A6 pp1-7 (1996)

31. I.S. Graham, The HTML Source Book. John Wiley, New York (1995)

32. Balachander Krishnamurthy, Jeffrey C. Mogul, David M. Kristol, "Key Differences between HTTP/1.0 and HTTP/1.1". In Proceeding of the Eight International World Wide Web Conference, Toronto, Canada, 11-14 May 1999 (1999)

33. Lu Qin, Lee Kin-Hong & Yao Jian. "Chinese Information Access Through Internet", In Proceeding of The World Conference of the WWW, Internet & Intranet 1997 (WebNet 97), Toronto, Canada, 1-5 November, 1997 (1997)

34. Peter Belesis, Identifying Installed Fonts, http://www.webreference.com/dhtml/column30, (2000)

35. Raggett Dave, Arnaud Le Hors, Jacobs Ian, HTML 4.01 Specification. 24 Dec 1999, W3C, USA (1999)

36. Chris Wilson, Philippe Le Hégaret, Vidur Apparao. Document Object Model (DOM) Level 2 Style Specification, W3C, USA (2000)

37. Elizabeth Castro, HTML 4 for the World Wide Web, Peachpit Press (2000)

38. Laurel. B, The Art of Human-Computer Interface Design, Addison Wesley (1992)

39. Edwin Hart, What you need to know about Processing and Rendering Multilingual Text. In Proceeding of Thirteenth International Unicode Conference, San Jose, California, 8-11 September 1998 TA4 pp1-21 (1998)

40. The TrueType Specification, http://www.truetype.demon.co.uk/ttspec.htm

41. The FreeType Homepage, http://www.freetype.org/

42. GD Graphic Library, http://www.boutell.com/gd/

43. libpng Home Page, http://www.libpng.org/pub/png/libpng.html

44. Info-zip Home Page, http://www.info-zip.org/pub/infozip/zlib/

45. Murray, William H., HTML 4.0: User's resource, Prentice Hall, (1998)

46. Castro, Elizabeth, HTML 4 for the World Wide Web, Peachpit Press, (2000)

47. Leong Kok Yong, Liu Hai, Oliver P. Wu, "Web Internationalization and Java Keyboard Input Methods", In Proceeding of The Internet Global Summit Conferences Conference (INET'98), Geneva, Switzerland, 21-24 July 1998 (1998)

48. Norbert Lindenberg. "Java Input Method Framework", In Proceeding of the Fifteenth International Unicode Conference, San Jose, California, 30 August – 2 September 1999, C16 pp1-13. (1999)

49. Tom McFarland, "Developing Internationalized Software with JDK 1.1", In Proceeding of the Thirteenth International Unicode Conference, San Jose, California, 8-11 September 1998, TC2 pp1-72. (1998)

50. M. Pong and Y. Zhang. Cxterm: A Chinese terminal emulator for the X Window system. Software – Practice and Experience, October 1992.

51. Robert L. Kruse. Trees and Graphs. Data Structures and Program Design. Second Edition. Prentice-Hall (1987)

52. Jian Yang, Kamalakar Karlapalem and Qing Li, "Algorithms for Materialized View Design in Data Warehousing Environment", Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97), August 25-29, 1997, Athens, Greece. (1997)

53. Leong Kok Yong, Liu Hai, Oliver P. Wu, "Java Input Method Engine", In Proceeding of The Seventh International World Wide Web Conference. Brisbane, Australia, 14-18 April, 1998 (1998)

54. Java 2 SDK, Standard Edition Documentation. Sun Microsystems (2000)

55. Core Javascript Guide Version 1.4. Netscape Communications Corporation (1998)

56. Reaz Hoque, Technology Evangelist. Java, Javascript And Plug-in Interaction Using Client-side LiveConnect. Netscape Corporation (1999)

57. John O'Conner, "Displaying Unicode with Java's Composite Fonts", In Proceeding of The Fifteenth International Unicode Conference, San Jose, California, 30 August – 2 September 1999, C17 pp1-70. (1999)

58. Richard Gillam, "Developing Global Application in Java", In Proceeding of The Sixteenth International Unicode Conference, Amsterdam, The Netherlands, 27-30 March 2000, TB3, pp1-88 (2000)

59. Norbert Lindenberg, Brian Beack. Java Developer Connection: Java Live! Internationalization, http://developer.java.sun.com/developer/community/chat/JavaLive/2000/jl0321.htmlMarch 21, 2000 (2000)

60. Raghavan N. Srinivas. Java security evolution and concepts, Part 3: Applet security, Java World, December 2000 (2000)

61. Laura Werner, "Unicode Text Searching in Java", In Proceeding of The Sixteenth International Unicode Conference, Amsterdam, The Netherlands, 27-30 March 2000, TB3, pp1-88 (2000)

62. Ka-Kit Hoi, Dik-Lun Lee, "Document Visualization on Small Display", In Proceeding of The Tenth International World Wide Web Conference, Hong Kong, 1-5 May 2001 (2001)

A. Appendix I

	9.6 Kbps	56 Kbps	128 Kbps	1.5 Mbps
50 CJK characters (1 KB)	0.83	0.14	0.06	0.01
100 CJK characters (3 KB)	2.50	0.43	0.19	0.02
200 CJK characters (5 KB)	4.17	0.71	0.31	0.03
500 CJK characters (9 KB)	7.50	1.29	0.56	0.05
1,000 CJK characters (15 KB)	12.50	2.14	0.94	0.08

Table A-1: Theoretical transmission time for retrieving Web pages from a Web server

	9.6 Kbps	56 Kbps	128 Kbps	1.5 Mbps
50 CJK characters (1 KB)	44.17	7.57	3.31	0.28
100 CJK characters (3 KB)	90.00	15.43	6.75	0.56
200 CJK characters (5 KB)	178.33	30.57	13.38	1.11
500 CJK characters (9 KB)	440.00	75.43	33.00	2.75
1,000 CJK characters (15 KB)	875.00	150.00	65.63	5.47

Table A-2: Theoretical transmission time for retrieving Web pages through Pelutech with conversion

	9.6Kbps	56Kbps	128Kbps	1.5Mbps
50 CJK characters (1KB)	1.67	0.29	0.13	0.01
100 CJK characters (3KB)	5.00	0.86	0.38	0.03
200 CJK characters (5KB)	8.33	1.43	0.63	0.05
500 CJK characters (9KB)	15.00	2.57	1.13	0.09
1,000 CJK characters (15KB)	25.00	4.29	1.88	0.16

Table A-3: Theoretical transmission time for retrieving Web pages through Pelutech without conversion

	9.6 Kbps	56 Kbps	1.5 Mbps
50 CJK characters (1 KB)	8	2	1
100 CJK characters (3 KB)	11	3	1
200 CJK characters (5 KB)	13	4	1
500 CJK characters (9 KB)	28	7	2
1,000 CJK characters (15 KB)	49	9	3

Table A-4: Actual Transmission Time for retrieving Web page from original server

	9.6 Kbps	56 Kbps	1.5 Mbps
50 CJK characters (1 KB)	63	10	2
100 CJK characters (3 KB)	131	23	3
200 CJK characters (5 KB)	253	43	5
500 CJK characters (9 KB)	ERROR	101	7
1,000 CJK characters (15 KB)	ERROR	186	10

Table A-5: Actual Transmission Time for retrieving Web page through Pelutech with conversion

	9.6 Kbps	56 Kbps	1.5 Mbps
50 CJK characters (1 KB)	11	4	2
100 CJK characters (3 KB)	19	6	2
200 CJK characters (5 KB)	26	8	3
500 CJK characters (9 KB)	55	10	4
1,000 CJK characters (15 KB)	75	13	5

Table A-6: Actual Transmission Time for retrieving Web page through Pelutech without conversion

	9.6 Kbps	56 Kbps	1.5 Mbps
50 CJK characters	10.00	3.10	2.00
100 CJK characters	15.60	4.20	2.00
200 CJK characters	18.60	5.00	2.00
500 CJK characters	30.60	8.50	3.40
1,000 CJK characters	53.60	10.60	5.60

Table A-7: Average Transmission Time for retrieving Web page from original server

	9.6 Kbps	56 Kbps	1.5 Mbps
50 CJK characters	59.30	12.50	2.50
100 CJK characters	128.60	24.30	3.00
200 CJK characters	255.30	40.60	4.90
500 CJK characters	ERROR	86.30	7.10
1,000 CJK characters	ERROR	162.30	11.20

Table A-8: Average Transmission Time for retrieving Web page through Pelutech with conversion

	9.6 Kbps	56 Kbps	1.5 Mbps
50 CJK characters	11.30	4.20	2.00
100 CJK characters	23.20	7.50	2.20
200 CJK characters	26.30	8.60	2.50
500 CJK characters	56.30	10.40	3.80
1,000 CJK characters	76.70	13.20	6.00

Table A-9: Average Transmission Time for retrieving Web page through Pelutech without conversion