Multilingual Input System for the Web — an Open Multimedia Approach of Keyboard and Handwriting Recognition for Chinese and Japanese

Marshall C. Ramsey
MIS Department
Karl Eller Graduate School of Management
University of Arizona
McClelland Hall 430W
Tucson, Arizona 85721
mramsey@bpa.arizona.edu
(520) 621-3927

Thian-Huat Ong
MIS Department
Karl Eller Graduate School of Management
University of Arizona
McClelland Hall 430W
Tucson, Arizona 85721
tong@bpa.arizona.edu
(520) 621-3927

Hsinchun Chen
Associate Professor
MIS Department
Karl Eller Graduate School of Management
University of Arizona
McClelland Hall 4302
Tucson, Arizona 85721
Visiting Research Scientist, NCSA
hchen@bpa.arizona.edu
(520) 621-4153

Abstract

The basic building block of a multilingual information retrieval system is the input system. Chinese and Japanese characters pose great challenges for the conventional 101-key alphabet-based keyboard, because they are radical-based and number in the thousands. This paper reviews the development of various approaches and then presents a framework and working demonstrations of Chinese and Japanese input methods implemented in Java, which allow open deployment over the web to any platform, The demo includes both popular keyboard input methods and neural network handwriting recognition using a mouse or pen. This framework is able to accommodate future extension to other input mediums and languages of interest.

1 Introduction

Many commercial and shareware products to support East Asian languages exist, but each works only on a specific platform. None is yet able to embrace the open standard of the web, which is destined to be the communication channel in all information systems. The web implementation using Java presented here will ensure its widespread deployment, but although Java provides multilingual internationalization support, it inherently assumes that each user already has an input system. For example, a computer in China or Japan will of course have an input mechanism for the local language but may not have input mechanisms for another language. This Java web implementation can ensure that all platforms will be supported.

As multilingual information retrieval becomes more and more important, people in different parts of the world continue to speak their own languages. While rapid growth of the Internet connecting every corner of the world promises a future of multilingual information systems and non-English speaking countries participate more and more in the Internet, information sources become more diverse in languages.

2 Various Input Methods

Input methods can be categorized by the input medium: keyboard, pen or mouse for handwriting, and dictation. Keyboard input was the first to develop because of simple implementation. However, the number of Chinese and Japanese characters presents a great challenge for 101-key English keyboard. There are about 48,200 Chinese characters according to a 1920 dictionary. Several standards have evolved over time and limited these to a smaller more commonly used set, such as Taiwan's BIG5 (about 13,300 characters used in Taiwan), GB (6,800 China), Japan's JIS (6,900 Japan), and their variations. The general approaches to using these can be divided into dictionary-lookup, phonetic, radical, and mnemonic. Details will be discussed in a later section.

Handwriting recognition is gaining grounds as an input method, and increased computing power and new

0-8186-8464-X/98 $10.00

input devices such as mouse and pen have become inexpensively available to the general public. Recent advancement in neural networks also has helped improve the accuracy of recognition rates. Using this method, users can input the characters naturally as if writing on a piece of paper. A program then attempts to map the character into the computer's internal code. This method is highly desirable because users can interact with the system with little or no training.

The last input medium is dictation, through which the system types as the user talks. This approach is very interesting, but voice dictation may create interference especially in an office environment. In addition, it requires such extensive hardware and sophisticated software that it will not be considered here.

3 Methodology

Different users have different needs, and they should be allowed to choose the input methods they prefer. For example, hand written methods novice users prefer can be considered too slow for advanced frequent users who may prefer quicker mnemonic input methods. By using the Object Oriented approach, our implementation framework as shown in Figure 1 allows multiple media (keyboard or pen/mouse) and multiple languages. The consistent design framework allows future extensions to incorporate other input media and to support other languages.

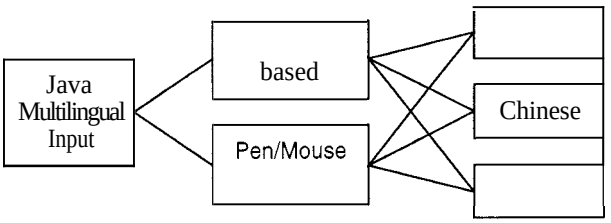


Figure 1: Multimedia multilingual input framework.

4 Keyboard Input Methods

Keyboard input methods can be divided into four categories: dictionary-lookup, phonetic, radical, and mnemonic. Dictionary-lookup works like a bilingual dictionary. The user enters an English word, and the system will return the corresponding Chinese or Japanese word. It is useful sometimes when the meaning of a particular word has been forgotten. For example, *tomorrow* will return as 明日. But generally, it is not helpful for daily uses because the dictionary would have to be very large to be comprehensive and many words do not have exact correspondence.

Phonetic-based input methods are one of the first methods to come into general use, but there are too many repetitions of characters with the same pronunciation. An extreme case might have up to **130** characters. For Chinese, there are two systems of phonetic symbols: China uses the romanized Pin Yin (拼音), while Taiwan uses the traditional 42-symbol Zhu Yin (注音). For example, "*ming2 ri4*" and "ㄇㄧㄥˊ ㄖˋ" both respectively represent *tomorrow* (明日), and people usually know one but not the other. The Japanese have long adopted a system of Romaji to write words in plain alphabet. For example, ***"myonichi"*** represents *tomorrow* (明日).

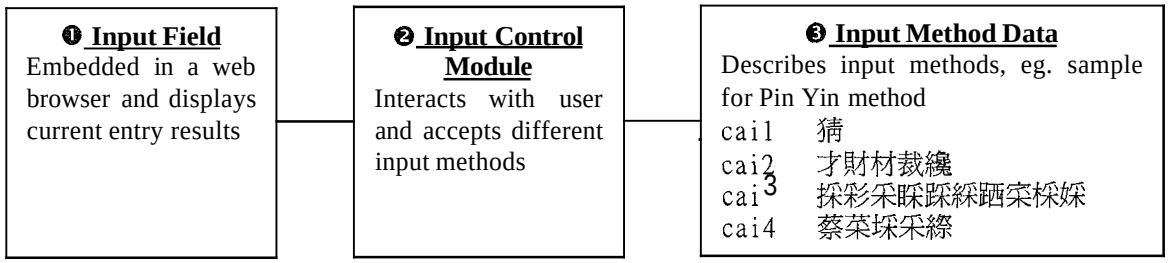
Radical-based input methods rely on a stroke sequence in which a character is written by hand. Certain keys are reserved to represent the basic radicals, and users enter these basic radicals in order reproduce the characters. However, some characters can take more than **30** radicals. Alternately, one can enter the number of radicals to initiate the system to look up all the characters with that number of radicals. A typical dictionary usually has a table of common radical components and subsequently divides the entries by the number of remaining radicals. However, the problem of repetitions also happens to this method.

Mnemonic input methods refer to those input methods that rely on a set of rules that are intended to minimize repetitions, in order to increase speed and maximize human associative memory. These methods usually parse characters into a particular sequence of keys according to radicals, components, or placements. There are more than 50 mnemonic methods, including the widely used Cang Jie, 4Corner, and many others. Users have recorded a speed over **300** characters per second using improved versions, but all mnemonic input methods require extensive training to attain top speed.

4.1 Implementation

The development of a Chinese keyboard input method involves a three-tier architecture as shown in Figure 2: Input Field, Input Control Module, and Input Method Data. The advantage is that a single input control module program can adapt to many different input methods.

The Input Field ❶ is embedded into a web browser, such as Internet Explorer or Netscape. Its main purpose is to be the liaison between the browser and the Input Control Module. It appears as a regular input field, but in fact it is a Java applet. If the browser is not able to display Chinese fonts in the applet, it will download a bitmap font from the server to display the Chinese characters properly. Through simple JavaScript, the browser can then use the input results from the applet to submit a regular CGI form query.



The Input Control Module ❷ is the central unit that allows a user to choose among different input methods available, takes keystrokes from the user, and then translates them into Chinese characters according to the input method selected. The user can also choose to enter English text, which is implemented as a trivial input method adhering to the general architecture.

Each individual input method is entirely captured in the Input Method Data ***O,*** which describes how each valid key sequence is mapped to one or more corresponding Chinese characters. The format is compatible with the human readable Textual Input Table (.tit) format used by CXTerm [3], a popular Chinese X-Term distributed with Unix. Additional .tit files are available on the Internet [3]. Since .tit files are stored in human readable text files, they are not space efficient, so an alternate more compact storage file format using a trie structure is also accepted by the Input Control Module.

4.2 Input Demostration

Figure *3* shows a demo, which is also available at **http://ai.bpa.arizona.edu/mlir/,** on how to use the keyboard input method Java applet with Netscape 4.0. The Input Field applet ❶ disguises itself as a typical input field which is part of the web query form. It is in the process of entering the word *ocean* 海洋 in Chinese.

When the user uses **a** mouse to click on the input field, it will bring up the Input Control Module ❷ as a separate window. Currently, it is now using the Pin Yin 拼音 (romanized pronunciation) input method. **As** the user keys in, it continuously updates the list of available choices matching the current key sequence. For example, the partial list shows characters with pronunciation *hai*.

The user can change the input method by clicking on the method name. Window ❸ shows a list of ten popular input methods, including Pin Yin, Cang Jie, 4Corner, and so on. The user can compile his/her own Data Input Methods or download others from the Internet.

4.3 Query Demostration

To demonstrate the use of this implementation as the front end interface to input Chinese characters to do multilingual information retrieval, the next demo in Figure 4 continues the previous demo and submit the query *ocean* 海洋 to Infoseek, a well known search engine. Since this is a generic word, Infoseek found 67,076 pages containing the word, and the first page is shown. To limit our search to Chinese documents, the search domain is further limited to Taiwan.

Interestingly, the first record is about discussion of El Nino in Taiwan and the second record is referring to the subcategory of the Ocean University in Yam, which is equivalent to Yahoo in Taiwan.

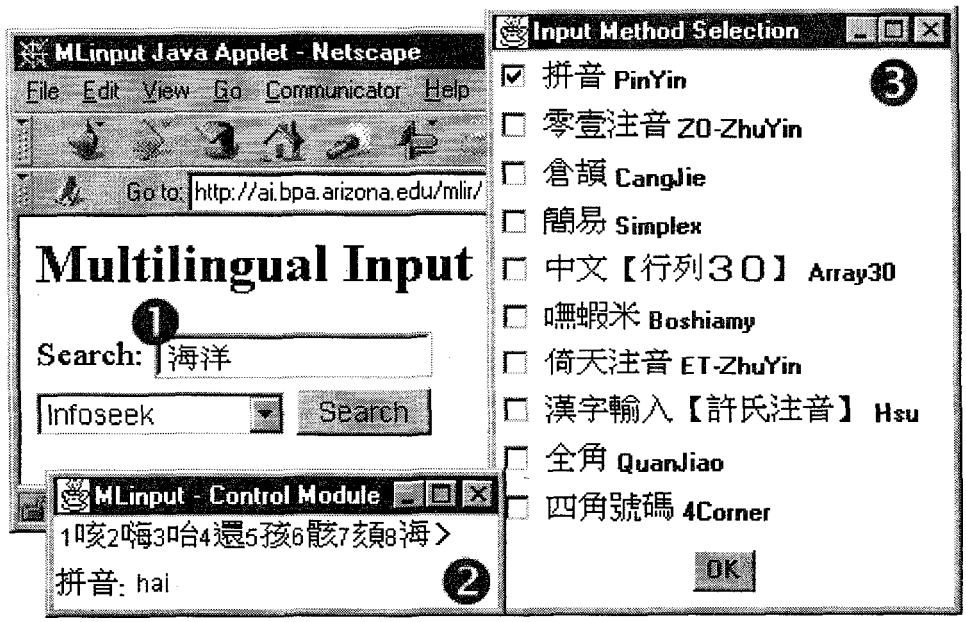


Figure **3:** Keyboard input method demo Java applet.

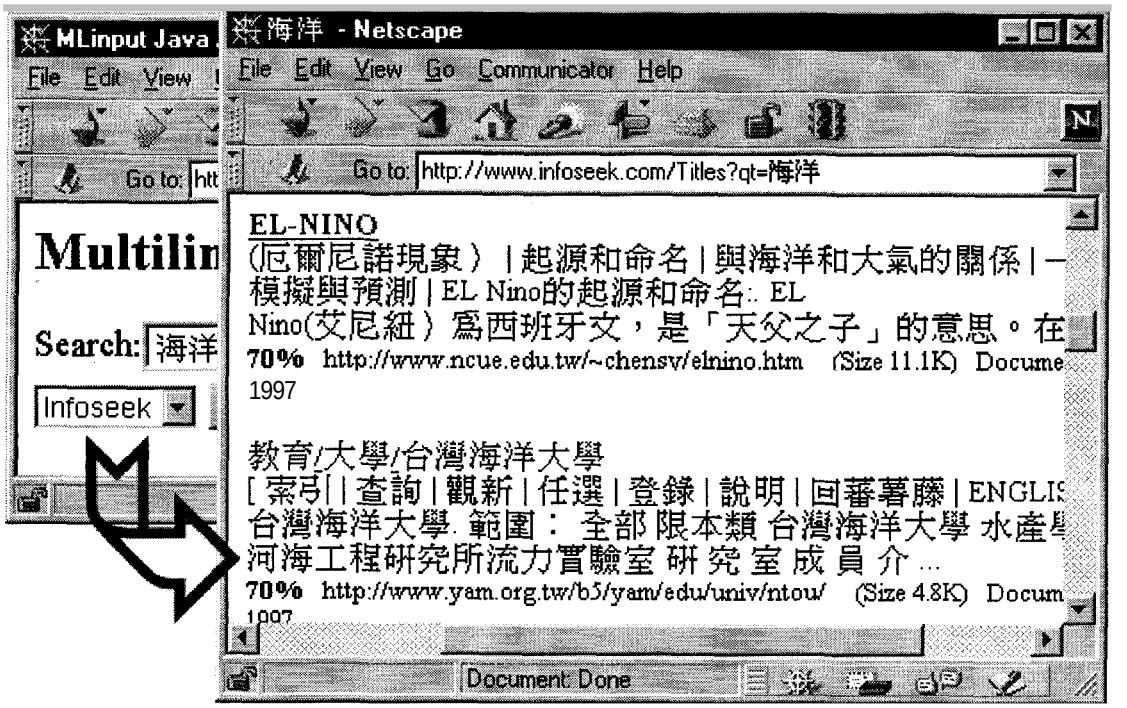


Figure 4: CGI query by keyboard input method

5 Handwriting Recognition

Research in handwriting recognition (HWR) is not new. Many researchers are or have been building such systems for decades. Nouboud and Plamondon provide an excellent survey of research in on-line written character recognition [2]. However, only recently has the technology been incorporated in popular consumer devices called Personal Digital Assistants (PDAs) such as the Apple Newton and 3com Pilot. Text is entered by drawing characters either in a special area or directly on the display. Both PDAs only recognize Roman alphanumeric and punctuation characters and do not process Japanese writing. The Pilot recognizes only character shorthand called Graffiti that must be learned before using it. Eventually, most PDAs will have handwritten input including those based on Microsoft's Windows CE for which ParaGraph Software already has released a HWR program called CalliGrapher.

The PDA run-time environment is similar to the Java Virtual Machine used to run Web applets. Both are relatively low memory, low processing power environments and require parsimonious programs. The Java Virtual Machine may be run on a wide range of platforms with varying processor speeds. Additionally, Java enabled Web browsers have security managers that limit the amount of memory applets can use. Our goal is to develop both Japanese and Chinese HWR systems that are not only accurate, but can also be run in this environment. In phase one of the research, we focused on Japanese character recognition of more than 2000 Chinese and native characters in common use.

To recognize Kanji, Chinese characters used for Japanese, we used the technology in JavaDict, a Japanese dictionary program written in Java that recognizes handwritten Kanji using a stroke index. The author, Todd David Rudick, claims a more than ninety percent recognition rate and, although not formally tested, our experience seems to support this. However, the program only recognizes Kanji and cannot decipher Kana, native Japanese characters mostly used to for inflection and foreign words. Moreover, JavaDict does not effectively distinguish low stroke count characters -- those with less than five strokes. Most Kana also have low stroke counts and are difficult to recognize with simple stroke indexing.

For our first phase, we tested a feed-forward back-propagation neural network for effectively and efficiently recognizing Kana. Also, we implemented the system in Java and used it as a front-end-processor (FEP) for Japanese search engines and dictionaries on the Web.

5.1 Sample Collection

We first created a character entry program to collect Kana writing samples from three different people, one native and two non-native Japanese speakers. The system consisted of a small handwriting entry window (i.e., about

200x200 pixels) and a text field for entering the Kana phonetically using the keyboard. Users entered a single character in the drawing area and typed the same character in the text field. In all, 107 Hiragana were entered and the data were saved to a file. **A** writing pad and stylus were used to draw the characters since they closely resemble the pen and paper normally used when writing. The 107 Hiragana and 132 Katakana character samples were collected from seven different people, five native and two non-native speakers.

5.2 Recognition by Neural Network

A three-layer feed-forward back-propagation neural network was then created and the data converted for input. Originally, each stroke was a set of points sampled from the mouse input and connected by line segments. Five points, including end-points were chosen at set intervals. From the five points, four X and four *Y* distances were calculated by subtracting each point's *X* and *Y* value from the first point's values. The values were normalized by first dividing each value by the character's maximum distance. This resulted in a value between -1 and 1 for each distance. The distances then had 1 added and the result was divided by 2. The final values were between 0 and 1.

Hiragana have a maximum of 9 strokes, which result in a total of 72 values (i.e., 9x4 X values and 9x4 *Y* values). Characters with less than 9 strokes had 0s appended. The resultant vectors were then used as inputs to the neural network.

The output layer consists of 107 nodes with each node representing a different Hiragana character. The network has 90 nodes in the hidden layer and utilizes a canonical sigmoid activation function. **A** learning rate of 0.35 without momentum was applied. We used three native and one non-native writing samples for training and two native samples for tuning.

5.3 Manual Stroke Indexing

As mentioned previously, Rudick utilized another approach to recognizing the characters in JavaDict that uses stroke indexing and filtering. He used 13 stroke tags as shown in Figure 5 to manually index about 2,000 Kanji. In his program, a character is drawn on the display and converted into a vector of strokes. Each of these strokes is represented by two vectors of X and *Y* points. The index file for all the characters with the same number of strokes **is** used to compute scores. The more closely the drawn strokes match the indexed character's stroke tags, the lower the score. When this has been completed, an ordered list of the top five matches, characters with the lowest scores, is returned.

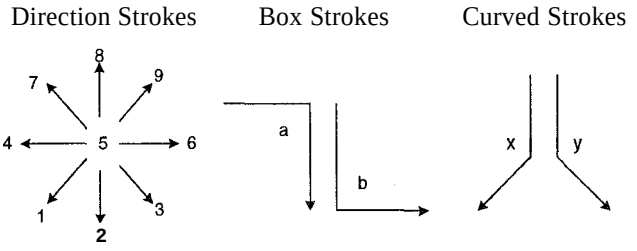


Figure 5: Stroke tags used for indexing.

During testing, Rudick added filters to characters that were not recognized correctly, using seven different filters that could be applied to any stroke in the character. His system required the filters to be applied in subtractive pairs (i.e., filter, - $filter_2$). The difference was then subtracted from the character's score from tags to compute the final score. The filters were then saved in the index file.

5.4 Automatic Stroke Indexing

Manually indexing characters is a time consuming process that we wanted to avoid. Instead, we created an automatic indexing program that uses the same function that scores the indexed characters to determine the index tags. For each stroke, we scored all 13 tags and chose the one with the lowest score. To better generalize, we used three different handwriting samples and chose the tag with the overall lowest score when applied to the three samples. Filters were also applied when characters were not properly recognized by the resultant indexes. The difference between the score of the correct character and the score of the top returned character was utilized to select a filter pair. The seven filters were applied to all the strokes in both characters and a pair was selected that had the minimum difference between the correct character's filter score and the top choice's filter score and was greater than the difference between their tag scores. If a filter combination matching the criteria was not found, then the difference between the correct character's and the top choice's filter scores only had to be greater than half the tag score differences. Thus, the correct character's score was guaranteed to be lower than the top choice's score in the next round. However, the correct character's score was not guaranteed to be the top choice since filters applied to other characters may have given them a lower score.

5.5 Results

After automatic indexing but without applying filters, recognition rates of 8% (top 1) and **33%** (top *5)* were obtained. When filters are applied the recognition rates jumped to 16% and 42%. The top rate of 42% is problematic since users would either have had to scroll through a long list to find the character they entered or not be able to select it at all. This poor performance was expected and suggested by Rudick.

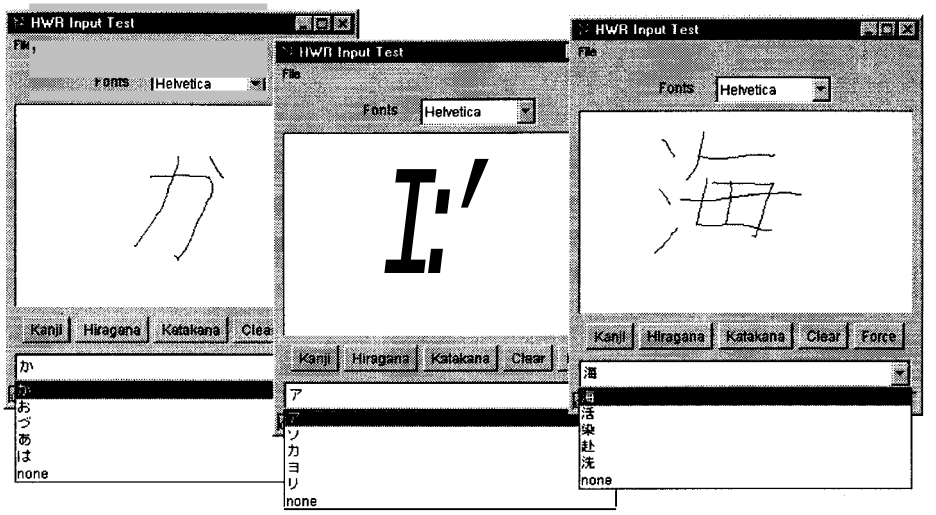


Figure 6: Handwriting entry windows with recognized characters.

The Hiragana neural network was originally trained for 1,500 epochs. The greatest recognition rate was 98% and was achieved at epoch 2,300. The network was then retrained using the same initial weights and stopped at epoch 2,300. The optimized network was then tested using unseen, non-native (first year student level) Hiragana input and had a recognition rate of 97%.

The Katakana neural network was also trained for 1,500 epochs. The best recognition rate of **96%** was reached at epoch 670. The network was retrained and stopped at this epoch. **A** recognition rate of 88% for unseen, non-native (first year student level) Katakana writing was achieved.

5.6 Demonstration

To demonstrate the recognition system, we created a Java applet that recognizes written Japanese input and inserts the characters into a Web form. Users wrote a character in the applet's drawing area and then pressed the button associated with the character type (i.e., Kanji, Hiragana, or Katakana) as shown in Figure 6. The top five matches were then shown sorted in a drop-down list. The character was inserted in the Web form when the user wrote another character or pressed the Force button. Figure 7 shows the entry of characters in a Web form connected to the Yahoo! Japan Web search engine. The results of the query are displayed in the foreground window with the query term in bold text.

An on-line demonstration is available at http://ai. bpa. arizona.edu/~mramsey/hwrkana2/hwr.html. Users will need to make sure that their browsers are Java Development Kit (JDK) 1.1 compliant and their system has a Japanese font.

5.7 Discussion

From the results, we conclude that our back-propagation neural network is an effective means for an on-line HWR system for both Hiragana and Katakana. From this we predict that the neural network will also improve the recognition rate for Kanji with few strokes. We are unsure if the neural networks recognition rate for high stroke count characters will be greater than automatic indexing results. However, since the number of input nodes is equal to eight times the maximum number of strokes and the number of output nodes equals the count of unique characters, the neural network approach currently requires too much memory and processing resources for convenient use in low resource environments.

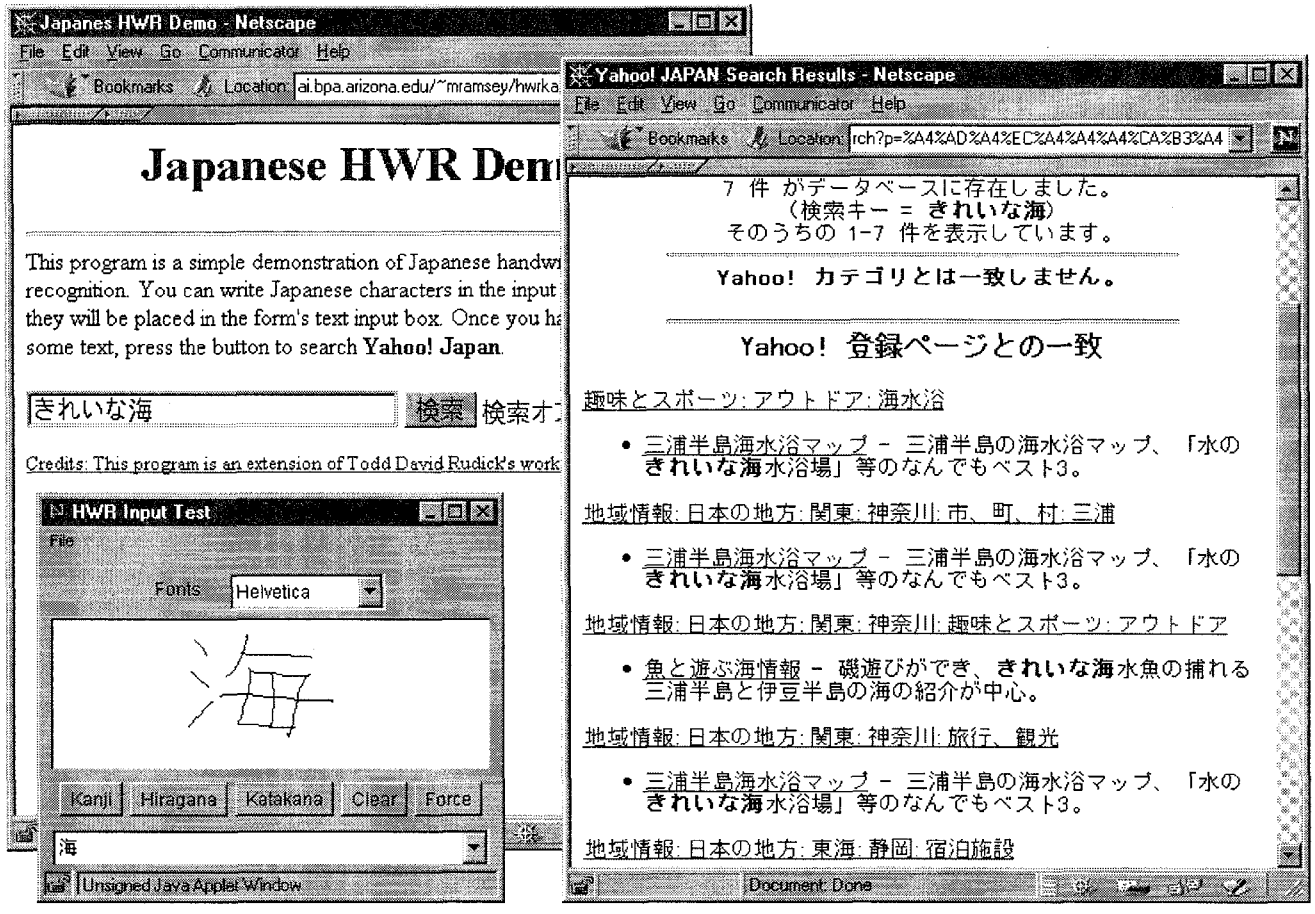


Figure7: Search form and results for Yahoo! Japan showing query entered by the Java HWR program.

6 Conclusion

The framework and working demos together have demonstrated a workable input system for different languages, which is a key building block in any multilingual information retrieval system. The demos showed a Java web implementation using Chinese and Japanese input via keyboard and handwriting recognition that can interoperate over all common platforms. The framework can be adapted to different needs of different users, including multiple input media such as keyboard, mouse and pen. Built upon the internationalization capability of Java, it can be extended to other languages in the future.

7 Acknowledgement

This project is supported by the following grants: NSF/DARPA/NASA Digital Library Initiative, IRI-9411318, 1994-1998 (B. Schatz, H. Chen, et. al, "Building the Interspace: Digital Library Infrastructure for a University Engineering Community") and DARPA Information Management Program, N66001-97-C-8535, 1997-2000 (B. Schatz, H. Chen, "The Interspace Prototype: An Analysis Environment for Semantic Interoperability").

We would also like to thank Bruce Schatz and Ron Larson for their comments and insight that have guided us in our research. Additionally, we wish to thank Kyoko Ramsey for her extraordinary effort in collecting Japanese data for the HWR system.

References

[1] H. Huang and C. D. Chen. Linux Chinese HOWTO English Version. *Linux Documentation Project.* http://www.sunsite.unc.edu/LDP/.

[2] F. Nouboud and R. Plamondon. On-line Recognition of Handprinted Characters: Survey and Beta Test. *Pattern Recognition*, 23(9):1031-1044, 1993.

[3] M. Pong and Y. Zhang. cxterm: a Chinese terminal emulator for the X Window system. *Software - Practice and Experience,* 22(10):809-26, October 1992.