A Free Tamil Keyboard Interface for Business and Personal Use

Panmozhi Vaayil - A Multilingual Indic Keyboard Interface
Abhinava Shivakumar, Akshay Rao, Arun S, A. G. Ramakrishnan
MILE Lab, Department of Electrical Engineering, Indian Institute of Science
Abhinav.zoso, u.akshay, sarun87, agrkrish@gmail.com

Abstract

A multilingual indic keyboard interface is an Input Method [1] that can be used to input text in various Indic languages like Tamil, Kannada, Telugu, Hindi, Gujarati, Marathi and Bengali. The input can follow the phonetic style making use of the standard QWERTY layout along with support for popular keyboard and typewriter layouts [1] (also known as soft layouts) of Indic languages using overlays. Indic-keyboards provides a simple and clean interface supporting multiple languages and multiple styles of input working on multiple platforms. XML based processing makes it possible to add new layouts or new languages on the fly. These features, along with the provision to change key maps in real time make this input method suitable for most, if not all text editing purposes. Since Unicode is used to represent text, the input method works with most applications. This is available for free download and free use by individuals or commercial organizations, on code.google.com under Apache 2.0 license.

Keywords-Input Method, IM, Indic, Localization, Internationalization (i18n), FOSS, Unicode, panmozhi vaayil, vishwa vaangmukha

Introduction

Input method editors or IMEs provide a way in which text can be input in a desired language. Traditionally, IMEs are used to input text in a language other than English. Latin based languages (English, German, French, Spanish, etc.) are represented by the combination of a limited set of characters. Because this set is relatively small, most languages have a one-to-one correspondence of a single character in the set to a given key on a keyboard. When it comes to East Asian languages (Chinese, Japanese, Korean, Vietnamese etc.) and Indic languages (Tamil, Hindi, Kannada, Bangla etc.), the number of key strokes to represent an akshara can be more than one, which makes using one-to-one character to key mapping impractical. To allow for users to input these characters, several input methods have been devised to create Input Method Editors. The term ***input method*** generally refers to a particular way to use the keyboard to input a particular language. The term ***input method editor*** refers to the actual program that allows an input method to be used.

Objective

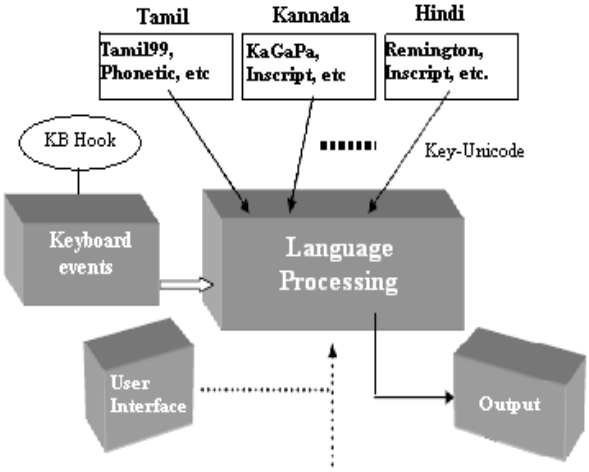
The focus has been to develop a multilingual input method editor for Indic languages. The interface should be minimalistic in nature providing options to configure and select various language layouts. Configurability is inclusive of addition of new layouts or languages and option to enable or disable the . Inputs can be based on popular keyboard layouts or using a phonetic style [2]. We call it Panmozhi

Vaayil in Tamil and Vishwa Vaangmukha: in Sanskrit, both meaning entrance for many languages. It is known by the generic name, Indic Keyboard IM in English and is available for download from http://code.google.com/p/indic-keyboards/

Motivation

Some of the main reasons for developing indic-keyboards are as follows:

1. To ease inputting of any Indian language under any platform.
2. To facilitate increased use and presence of Indian languages on the computer and internet.
3. To provide a free with an unrestricted license.



4. Phonetic as well as popular layout support in a single package.
5. Need for a unified multiplatform input method.
6. Ease of configurability and customizability.

Figure 1 shows the system architecture of Panmozhi Vaayil showing the various modules and their interaction.

Existing works and what they offer

A good amount of effort has already gone into the development of easy, flexible input methods. Some popular ones are:

- **Baraha IME –** Provides phonetic support for a fixed number of languages and designed for use on Microsoft Windows platform [3].
- **Aksharamala –** Similar to BarahaIME with support for Microsoft Windows [4].
- **Smart Common Input Method (SCIM) –** Designed to work on Linux with phonetic style of input [5].

What indic-keyboards (Panmozhi Vaayil) offers

- No installation hassles.
- Phonetic as well as popular keyboard layouts.
- Dynamic module enabling the addition of new keyboard layouts by even users.
- Both on Linux platform and Microsoft Windows.
- Phonetic key maps can be changed to meet user's requirements.

- Open source.
- Available under Apache 2.0 License, which means even commercial companies can use our code to develop products, after acknowledging us.

Design

The design can be broadly categorized into the following modules:

- User interface and the shell extension.
- Capturing the keyboard events.
- XML based Unicode processing.
- Rendering the Unicode.

The following block diagram shows the architecture:

User Interface and Shell Extension

The User interface is a shell extension which sits in the system tray/Notification area. The main purpose of this is to allow users to interact with the input method. This mainly involves selection of the language and the particular keyboard layout. It also helps in enabling and disabling the input method, accessing help and to display the image of the keyboard layout currently selected. Apart from these, the menu also has provision for addition of new keyboard layouts.

Capturing the Keyboard Events

The input method is designed to operate globally. That is, once the input method is enabled, further key strokes will result in characters of the particular language selected being rendered system wide. This requires the capture of the key presses system wide across all processes. A keyboard hook installed in the kernel space will enable this. This module is, therefore, platform specific.

XML Based Unicode Processing

Finite Automata exists for each language (for every keyboard layout). The finite automata has been designed as XML files, where every XML file corresponds to a keyboard layout. XML based processing makes it possible to add new layouts or new languages dynamically. The XML file has a pattern, which corresponds to the input key(s) pressed. The input pattern is matched and required processing is carried out to see if the pattern matched is a vowel or a consonant. For the input pattern, a sequence of Unicode(s) is returned. The structure of the XML file is as shown below:

```
<pattern>
        <char>A</char>
        <unicode>0C86</unicode>
        <consonant>0</consonant>
        <uni2>0CBE</uni2>
</pattern>
```

The above XML block indicates that for the key press “A”, the corresponding Unicode is 0C86. The consonant tag tells us whether the key is a vowel or a consonant. In case it is a vowel, a second tag indicates the Unicode of the associated dependent vowel (if any).

Two algorithms have been designed, one for phonetic style of input and the other for keyboard layouts. Both the algorithms are generic, i.e. same algorithm is used for keyboard layouts of all languages and one algorithm for phonetic input in any language.

The XML key maps can be changed on the fly and the changes are reflected instantly.

Unicode Rendering.

Once the key is pressed, simple grammar rules are applied to determine whether the output has to be a consonant, an independent vowel or a dependent vowel. The XML file is parsed and the corresponding Unicode is fetched. The Unicode is sent back to the process, where the keypress event took place and is rendered if any editable text area is present. The rendering of Unicode is platform specific.

Implementation

The following tools and languages have been used to implement the input method:

Java SE ***– The main language processing module has been implemented using Java SE. This has enabled easy portability and up to 80% of the code has remained common across platforms.***

Eclipse SWT ***– Used to implement the user interface. Eclipse SWT uses Java SWT is preferred over other toolkits to get a native look and feel..***

XML ***– As described previously, finite automata exists for every language (layout) and XML has been used to design it. Simple API for XML Parsing (SAX) has been used to parse the XML.***

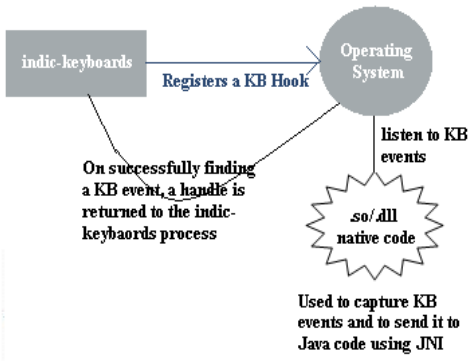
Win32 libraries ***: The Windows API, is Microsoft's core set of application programming interfaces (APIs) available for the Microsoft Windows platform. Almost all Windows programs interact with the Windows API. Some examples are SAPI, Tablet PC SDK, Microsoft Surface etc. Platform specific portions of the input method has been implemented to run on Microsoft Windows variants using the Microsoft Win32 libraries. Both keystroke capturing and Unicode rendering have been accomplished using Win32 libraries.***

Steps :

a. Syshook.dll : Install a keyboard hook in the operating system. The hook is set up for the keyboard to listen to key presses. The Windows API used is **SetWindowsHookEx()** and the library accessed is user32.dll. (See Fig. 2)

b. opChars.dll : Responsible for putting the character on to the current active window. Sends a message to the input event queue using the Windows API **SendInput()**. The library accessed is user32.dll

Fig. 2. JNI-Native code and keyboard hook procedure



Evdev ***- Also known as the user input device event interface/Linux USB subsystem. Used to capture keystrokes in GNU/Linux. This provides a clean interface to handle input devices like keyboard, mouse, joystick etc. in the userspace in Linux. This involves the following things :***

a. Open file descriptor to the input device using the **open()** API with suitable parameters. Use **ioctl()** to identify the device.

b. Using the open file descriptor, continuously read bytes from the input device (can be key press/release, mouse movements) using the **read()** API

Xlib ***– Used for Unicode rendering in GNU/Linux.***

Steps :

a. Identify the current active window using **XGetInputFocus()**

b. Make the window listen to all keypress events using **XSelectInput()**

c. Using the keycodes obtained for every keypress/release event from evdev, using a mapping table to map the keycode to the keysym. Output the Unicode to the active window using **XSendEvent()** API

Java Native Interface ***– Also known as JNI in short. The JNI enables the integration of code written in the Java programming language with code written in other languages such as C and C++. The write once, run anywhere concept arises from the fact that Java acts as an abstraction layer on top of the native implementation. All the API java provides have been natively implemented and the Java code allows the same APIs to be used across platforms.***

The native code is usually packaged as a DLL or a Shared Object. The Java method which accesses the native code is created with a keyword "native". Header files need to be created for the classes which contain these methods. At run-time, java code interacts with the native libraries using predefined interfaces. The native methods can also call Java methods. This mechanism is known as **JNI callback**.

Languages and Layouts Supported

TABLE I - *LANGUAGES AND LAYOUTS SUPPORTED*

Language	Phonetic	Layouts
Hindi	Yes	Inscript Remington
Kannada	Yes	Inscript KaGaPa
Tamil	Yes	Inscript Tamil99 Remington
Telugu	Yes	Inscript
Gujarati	Yes	Inscript
Marathi	Yes	Inscript Remington
Bengali	No	Inscript
Malayalam	No	Inscript
Oriya	No	Inscript
Gurmukhi	No	Inscript

Currently, the input method supports 10 Indian languages namely Tamil, Telugu, Kannada, Malayalam, Hindi, Bengali, Gurmukhi, Oriya, Gujarati and Marathi. The different keyboard layouts currently supported in these languages are listed in Table I.

An easy-to-use user interface has been provided to add new layouts which are Inscript like. Additional phonetic or other layouts can be added based on the existing layouts by creating new XML files and following the prescribed structure. Existing layouts can be changed/customized to suit the users' needs. In phonetic layouts, a single key press to vowel mapping is used to ensure lesser key presses for the completion of the CV combination. Ex : k (க்) + Y (ை) = கை instead of k + ae/ai.

Performance

The input method is multithreaded and the following runtime statistics have been obtained. Java Monitoring and Management console has been used to profile.

- Average Heap Memory usage : 4.0 MB (maximum : 5.0 MB)
- CPU usage : 0.2% – 0.3%
- Garbage Collector: Average time for one sweep – 0.05s. Average heap space freed up – 1 MB
- Number of threads: Peak – 15. Average live threads – 13 (2 threads are spawned by the input method)

Conclusion

In conclusion, the project has been an attempt to providing a very dynamic input method editor. The flexibility of adding new Indic languages on the fly, modification of the existing layouts, changing the keypress - Unicode input combination for phonetic input and a host of many other features makes for very easy to use . The main focus has been on flexibility; ease of use and to keep things to a minimum. Keeping that in mind, we have abstained from touching or modifying any system files as well as relieving the user of all installation hassles. Performance wise, the input method is fast and very light on system resources. From the user's perspective, all one needs to do is run it. This also means the user can run the through a pen drive, CD, DVD, hard disk or any portable media. The being open source and licensed under the Apache 2.0 License, developers and users alike can modify, recompile, or rewrite the entire source and can also make these appendages closed source. Apache 2.0 license also allows developers to sell the modified code. All in all, a very dynamic, flexible, easy to use, clean, unrestrictive input method has been designed, which is multiplatform and multilingual.

References

[1] Russ Rolfe. "What is an IME (Input Method Editor) and how do I use it?", Microsoft Global Development and Computing Portal, July 15, 2003

[2] A.G. Ramakrishnan, Abhinava Shivakumar, Akshay Rao, Arun S. "Indic-Keyboards – A Multilingual Indic Keyboard Interface", TIC 2009 Conference Book, p. 130-132, INFITT 2009.

[3] Baraha - Free Indian Language (htttp://www.baraha.com)

[4] Aksharamala (http://w*ww.**aksharamala**.com/*)

[5] *Smart Common Input Method* (*SCIM*) (http://www.scim-im.org/)

Authors

Abhinava Shivakumar, ***Project Staff, MILE Lab, Department of Electrical Engineering, Indian Institute of Science, Bangalore. abhinav@mile.ee.iisc.ernet.in***

Akshay Rao, ***Project Staff, MILE Lab, Department of Electrical Engineering, Indian Institute of Science, Bangalore. u.akshay@gmail.com***

Arun S., ***Project Staff, MILE Lab, Department of Electrical Engineering, Indian Institute of Science, Bangalore. sarun87@gmail.com***

A. G. Ramakrishnan, ***Professor, Department of Electrical Engineering, Indian Institute of Science, Bangalore. ramkiag@ee.iisc.ernet.in***