Panmozhi Vaayil - A Multilingual Indic Keyboard Interface for Business and Personal Use

Shiva Kumar HR, Abhinava Shivakumar, Akshay Rao, Arun S, Ramakrishnan AG

MILE Lab, Dept of Electrical Engineering, Indian Institute of Science, Bangalore

Abstract. A multilingual Indic keyboard interface is an Input Method that can be used to input text in any Indic language. The input can follow the phonetic style making use of the standard QWERTY layout along with support for popular keyboard and typewriter layouts [1] 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.

1 Introduction

Input method editors (IME) provide a way in which text can be input in a desired language other than English. Latin based languages are represented by the combination of a limited set of characters, and most languages have a one-to-one correspondence of each character 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.

2 Objective

The focus has been to develop an Indic multilingual input method editor, with a minimalistic interface providing options to configure and select various language layouts. Configurability is inclusive of addition of new layouts or languages. 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/

3 Motivation

Some of the main reasons for developing Indic-keyboards are:

1. To ease inputting of any Indian language under any platform.
2. To facilitate increased use of Indian languages on the computer and internet.
3. To provide a free interface with an unrestricted license.
4. Support phonetic and other popular layouts in a single package.
5. Need for a unified multiplatform input method.
6. Ease of configurability and customizability.

3.1 Existing works and what they offer

Some of the existing, popular, easy, flexible input methods 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 MS Windows [4].
- **Smart Common Input Method (SCIM)** – Designed to work on Linux with phonetic style of input [5].

3.2 What Indic-keyboards (Panmozhi Vaayil) offers

The distinguishing features of our input method editor are:

- Phonetic as well as popular keyboard layouts.
- Dynamic module enabling the addition of new keyboard layouts by users.
- Both on Linux platform and Microsoft Windows.
- Phonetic key maps can be changed to meet user's requirements.
- No installation hassles.
- Open source.
- Available under Apache 2.0 License, which means even commercial companies can use our code to develop products, after acknowledging us.

4 Design

Figure 1 shows the architecture. The design can be broadly categorized into the following modules:

(1) User interface and the shell extension. (2) Capturing the keyboard events. (3) XML based Unicode processing. (4) Rendering the Unicode.

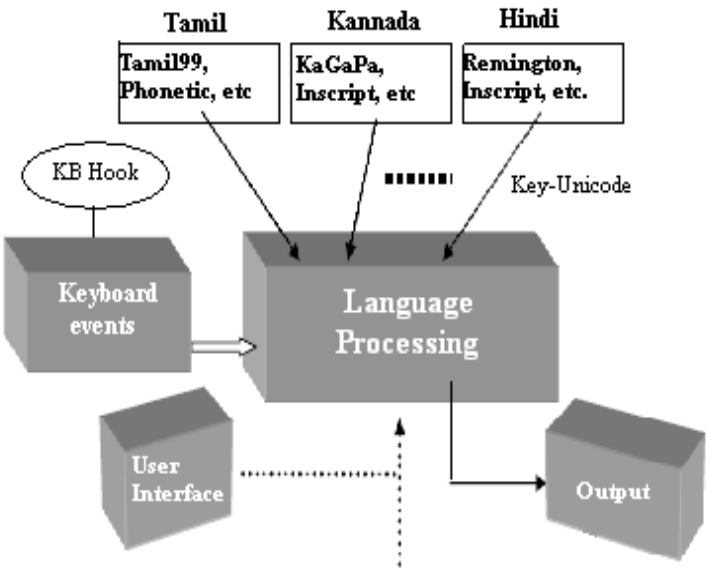

Fig. 1. System architecture of Panmozhi Vaayil showing the modules and their interaction.

4.1 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.

4.2 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.

4.3 XML Based Unicode Processing

Finite Automata exists for each language and for every keyboard layout. It has been designed as XML files, where every XML file corresponds to a kBD layout. XML based processing makes it possible to add new layouts or new languages dynamically. The input key pattern is matched with the XML file 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 follows:

```
<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 it is a vowel or consonant. If it is a vowel, a second tag gives the Unicode of the associated dependent vowel (if any).

Two algorithms have been designed, one for phonetic style input and the other for keyboard layouts. Both 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.

4.4 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.

5 Implementation

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

1)**Java SE** – This is used to implement the main language processing module to get easy portability. Up to 80% of the code has remained common across platforms.
2)Eclipse SWT – Used to implement the user interface. Eclipse SWT, which uses Java SWT, is preferred over other toolkits to get a native look and feel..
3)XML – 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.
4)Win32 libraries: Windows API, is Microsoft's core set of application programming interfaces (APIs) available for MS Windows platform. Almost all Windows programs (eg. SAPI, Tablet PC SDK, Microsoft Surface) interact with the Windows API. Platform specific portions have 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. The steps involved are:

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

5)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 keyboard, mouse, joystick etc. in the userspace in Linux. This involves the following:

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.

6)Xlib – Used for Unicode rendering in GNU/Linux. The steps involved are:

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.

7)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**.

6 Performance and Conclusion

The languages and keyboard layouts currently supported 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 user's 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 ( ) + Y ( ) = instead of k + ae/ai.

The input method is multithreaded and the following runtime statistics have been obtained. Java Monitoring and Management console has been used to profile.
(1) Average Heap Memory usage : 4.0 MB (maximum : 5.0 MB). (2) CPU usage : 0.2% – 0.3% (3) Garbage Collector: Average time for one sweep – 0.05s. Average heap space freed up – 1 MB. (4) Number of threads: Peak – 15. Average live threads – 13 (2 threads are spawned by the input method).

The flexibility of adding new Indic languages on the fly, modification of the existing layouts, changing the keypress - Unicode input combination for phonetic

input makes it easy to use. Thus, we have abstained from modifying any system files and relieved the user of all installation hassles. It is fast and light on system resources. Th user can run it through a pen drive, CD, DVD, hard disk or any portable media. Being open source and licensed under the Apache 2.0 License, developers and users alike can modify, recompile, or rewrite the source and can also make these appendages closed source. The license also allows developers to sell the modified code. Thus, a dynamic, flexible, easy to use, multiplatform, multilingual,.clean, unrestrictive input method has been designed.

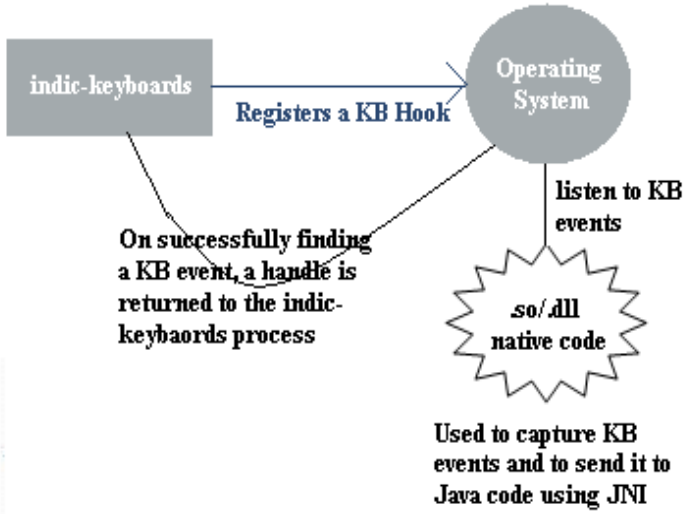

Fig. 2. JNI-Native code and keyboard hook procedure

Table I. Languages and layouts currently supported by Panmozhi Vaayil.

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

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. Baraha - Free Indian Language (htttp://www.baraha.com)
3. Aksharamala (http://w*ww.aksharamala.com/*)
4. *Smart Common Input Method* (*SCIM*) (http://www.scim-im.org/)