Creating Soft Input Methods for natively Unsupported Languages in Android Operating System

Tharindu Amila Perera,
Department of Computer Science and Engineering, University of Moratuwa
Sri Lanka
amilastbmmv@gmail.com

***Abstract*—The need for native language support for any hand-held communication device is a must. But Android operating systems do not give much freedom when altering system fonts which require many advanced steps that needs to be carried out by a user. The proposed solution is for developers of applications who use native languages to create their own in-app input methods and rendering.**

I.INTRODUCTION

If you have tried using any IME (Input Method) that supports languages not in the Android operating system there is the issue of not being able to view the characters of the keyboard. And the UI views of the text may not render the characters of the language if the support has not being built into the application being used.

And if you wanted any keyboard that is not currently in the system you will need to download and separately install the keyboard in need and then setup the system to use that keyboard as the default input method.

The alternative proposed to this is to develop the keyboard needed as a built-in feature of the application. This will give the convenience of having the needed functionalities right after installing the application that is needed.

II. RELATEDWORK

Android is an operating system based on the Linux kernel [1]. The operating system has been developed primarily for touch screen phones, tablets and computers. Android is now owned by Google and developed by the company as well.

Android is open source and Google releases the code under the Apache License [1]. This allows developers to freely alter and freely distribute the Android System.

There are only 3 fonts available as part of Android; normal (Droid Sans), serif (Droid Serif), and monospace (Droid Sans Mono). Android uses the system fonts to render the keyboard view of an input method limiting the options for a developer.

This does not allow a developer to display native language characters in the view in a normal and convenient manner.

III. D E S I G N AND IMPLEMENTATION

The main feature of this proposed solution is to create built in input methods that is customized for the application rather than create a common system IME which greatly reduces the flexibility needed. The design targets to reduce the effect of adding the new input method from disrupting the normal functionality of the application. So the design is completely separated from the application objects and only adds a view component to the UI.

The proposed method uses the views and UI libraries that are designed for creating separate IMEs. The view mainly used would be the keyboard view which will create the UI of the keyboard by using a XML file (Fig 1) which has to be defined according to the keyboard needs.

To overcome the issue with not being able to show non system font characters can be solved using gif

```
<?xml version="1.0" encoding="UTF-8"?>
<Keyboard xmlns:android="http://schemas.android.com/apk/res/android"
    android:keyHeight="10%p"
    android:keyWidth="9%p" >

    <Row>
        <Key
            android:codes="3540"
            android:keyEdgeFlags="left"
            android:keyIcon="@drawable/p3540" />
        <Key
            android:codes="3461"
            android:keyIcon="@drawable/p3461" />
        <Key
            android:codes="3536"
            android:keyIcon="@drawable/p3536" />
        <Key
            android:codes="3515"
            android:keyIcon="@drawable/p3515" />
        <Key
            android:codes="3473"
            android:keyIcon="@drawable/p3473" />
        <Key
            android:codes="3524"
            android:keyIcon="@drawable/p3524" />
        <Key
            android:codes="3512"
            android:keyIcon="@drawable/p3512" />
        <Key
            android:codes="3523"
            android:keyIcon="@drawable/p3523" />
        --
```

Fig. 1 Keyboard Layout Definition

pictures of the characters used as the key symbol to be shown. This method will fail if gifs are not created for separate resolutions. This is the only method to show the characters of a font that is needed by an application on a keyboard view which is not a system font. To create the fonts, a java program can be used that will reduce the effort needed and formatting of the pictures.

To handle input events (Fig. 2) a separate class extended from the OnKeyboardActionListener class is used. This class is also used in the normal IME creation thus reducing the need of implementing the keyboard functionality from the scratch.

To implement a separate keyboard for an application, adding a keyboard view holder in the existing UI of the application is needed.

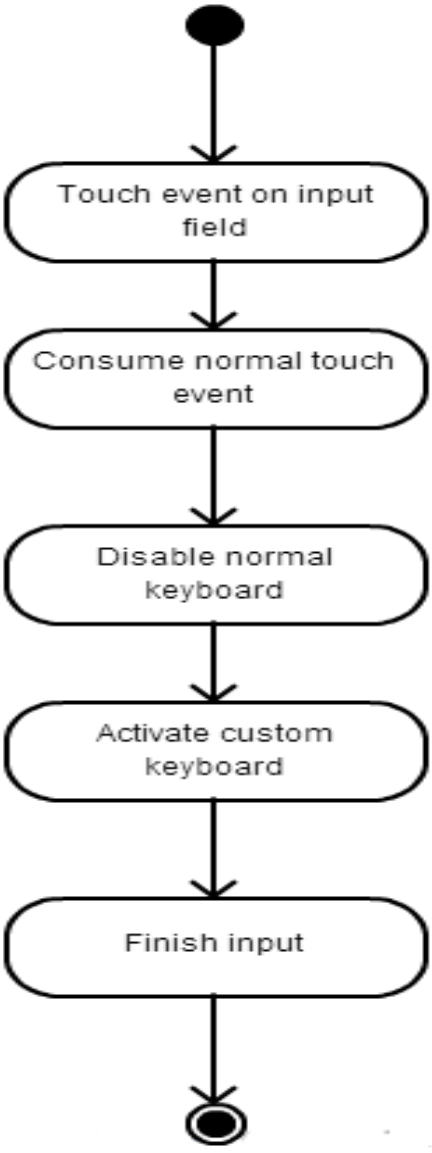

Fig. 2 User input work flow

The other classes of the keyboard do not have any effect on the functionality of the application. This allows code reuse that will allow for very flexible input method creation for applications.

The event handler will not disrupt the functionality of the existing application because the keyboard that is added into the application will be viewed as the normal input method of the Android operating system. This means that the addition to the application is completely separate from the functionality of that application. To implement the event handler all keys and its functionality will be bind through the values sent by the key according to the keyboard XML definition. The event handling class then maps the function according to the received event code. Default keys like new line and shift have specific codes and using them will ensure special events fired by the operating system will occur without breaking them.

In addition to the main requirement this method has allowed adding a custom suggestion view to give better usability to the keyboard. This was achieved by adding another view named a candidate view (Fig. 3) and supporting suggestions through natively supported SQLite database. This allowed better performance with virtually no hanging time of the keyboard. To further ensure better performance, retrieval of suggestions was done via an AsyncTask which was a separate thread running alongside the main application thread.

Fig. 3 Suggestion View

Fig. 4 Developed Keyboard View

Fig. 5 Completed Application View

IV. RE S U L T S

The method described above has been successfully integrated into a twitter client. The client named Kichibichiya had Sinhala rendering support to view tweets made using Sinhala but no method of creating Sinhala tweets. For this a Sinhala keyboard was required in the standard Wijesekara layout (Fig. 4).Including the keyboard view and the event handling classes, the new Sinhala keyboard was added to the arsenal of the application with relative ease and success. The UI of the application did not change instead the new input method was shown to give the input (Fig. 5).

V. ACKNOWLEDGMENT

I acknowledge the fact that the initial idea for this project was given by Mr. Nisansa Dilushan de Silva for whom I must thank for. Then I must equally thank Mr. Pahan Sarathchandra who is the developer and owner of the project Kichibichiya which is an open source project and gave invaluable advices and shared his vast domain knowledge on Android which made it possible to complete this project.

REFERENCES

[1] "*Android Overview*". Open Handset Alliance. Retrieved 2012-02-15.

[2] "*Android Developers*" [Online] Available: http://developer.android.com/

[3] "Android development: Custom keyboard" [Online] Available: http://www.fampennings.nl/maarten/android/09keyboard/index.htm

[4] "Rendering of Unicode Sinhala Characters" [Online] Available: http://www.ucsc.cmb.ac.lk/ltrl/publications/uni_sin_rndr.pdf