Indian Language Support for X-Window System

Authors:

Anitha Nalluri, Bala Saraswathi A., Bharathi S., Hema A. Murthy, Patricia J., Ranbir Singh S., Timothy A. Gonsalves, Vidhya M.S. and Viveka Nathan K.

TeNeT Group, Department of Computer Science and Engineering,

Indian Institute of Technology, Madras, Chennai - 600 036

E-mail: *indlinux@tenet.res.in*

Website: *http://www.tenet.res.in/Donlab/Indlinux*

Abstract

English is the medium of interaction in most of the widely available software today. In a country like India, wherein, an overwhelming majority of the population does not know English, availability of affordable native language software will play a crucial role in the development process.

For content generation in a particular language, operating system and the programming environment dependent input and display mechanisms need to be in place for that language. This paper addresses the process of providing Native Language Support for the X-Window System.

Variable-width characters, vowel modifiers and consonant clusters, and internationalization and localization of applications and compilers, have been the major issues that have been addressed in this paper. The X Window System based solution involves modifications to the X-library. Also a suite of applications has been adapted to support Indian languages under X-Window.

1. Introduction

Almost all widely available software today is written and documented in English. In a country like India, availability of affordable native language software will play a crucial role in the process of taking the benefits of the "information revolution"[Kenistion, 1998].

Section 2 explains about the multilingual support on X. Section 3 deals with issues related to Native Language Support and section 4 contains the proposed solution. Section 5 lists the some localized applications.

1.

Overview of Native Language Support(NLS)

1. Locale

Each country or language has its own set of native attributes like the country's cultural conventions, language-specific scripts (fonts), format of date and time, representation of numbers, currency-symbols etc. The formal description of these attributes together with associated translations targeted to a native language, constitute the *Locale* for the particular language or country.

2. Internationalization and Localization

Internationalization refers to the process by which a package is made aware of and is enabled to support multiple languages. This is a *generalization process*, by which the programs use specific ways of doing user-interaction and other locale-specific functions.

Localization provides the necessary information specific to a language or country to an internationalized package. This is a *particularization* process by which generic methods already implemented in an internationalized package are customized for a particular language or country.

2.

Scope of the paper

Encoding, font and display supports, that are largely dependent on OS and programming environment, are required for content generation in a particular language. This paper addresses these aspects for the X-Window System (X).

Developing a NLS at lower level is preferred, so that, all the applications running on top of it will inherit the interface, with no or minimal modification. The rest of this section gives an overview of certain definitions and scope of this paper.

2. Multilingual support on X-Window

X offers high flexibility for customization of the keyboard-input-display pipeline. The detailed mechanisms for the keyboard and display handling for X are given in the subsequent sections.

1. Keyboard Input Mechanism

The X server generates a keyPress event, when a key is pressed and a keyRelease event, when the key is released. Under X, the keyboard gets attached to the window or the sub-window, which has the focus and all the events are directed to it. The keyboard module is broken into two layers: server-specific codes (called *keycodes*), which represent the physical keys, and server-independent symbols (called *keysyms*), which represent the letters or words that appear on the keys.

The keycode is translated to meaningful character(s) by a two-step process:

1. Translating the keycode to a symbolic name, known as keysym.
2. Converting the keysym to an ASCII text string.

Appropriate utilities are available in X to create appropriate mappings.

1. Display Mechanism

Unlike text display terminals, X display, can show text in varying sizes and shapes. The X retrieves the glyphs from a font by indexing based on character code. There are mainly two types of font: bitmap based and outline or curve based fonts. True Type Fonts (TTF) are widely regarded as the best outline based font for low-resolution devices like X display.

1. Issues in providing Native Language Support

The various issues involved in providing NLS for the X are discussed in this section.

1. Character Width - Cursor Positioning Problem

Unlike English, in Indian languages, the mean deviation of width of characters varies largely and hence, the aesthetic appeal of characters is affected. For example, a character ண in Tamil, cannot be legibly fit in an 8-pixel width.

Even if scalable fonts are used, the variable width of the glyphs has an implication in the X, namely the cursor positioning. In the display mechanism, no history is maintained about the position of the cursor in the X-library (Xlib). Due to the variable width of the characters, the cursor positioning is to be calculated and sent to the application for Indian Languages.

2. Consonant and Vowel Clusters

Another feature in Indian languages is the concept of consonant and vowel cluster formations.

Consonant-vowel clusters of Indian languages result in non-trivial modified versions of the consonant.

Besides, the glyph ordering is also different in different languages. For example, consider the vowel modifier sounding like the English character *'e'*, applied to the consonant sounding: 'ka'. In Hindi, this takes the form: क +ि = कि ; But in Tamil it takes the form கி . The vowel modifier sounding like the English character '*A*' applied to க gives கா . In some cases, the consonant may get sandwiched between the components of the vowel modifier. For example, க (*Ka*) + vowel 'O' = கொ . Editing operations also need to be taken care of, while working with vowel modifiers. For example, if BkSpace is pressed at a character கொ it should give க , and not கெ . A cursor-positioning request, to go to the next column, should place the cursor after கொ , and not after கெ .

2. Proposed Solution

The focus of this work has been to provide a unified approach to address these problems across all languages that require variable width font and have the concept of modifiers. The interface allows co-existence of English and multiple native languages as well.

The requirements of an ideal solution are:

1. A facility to switch between different keyboard layouts.

2. Generic solution i.e., it should be language-independent.

3. The solution should address all the issues specified in Section 3.

4. The support should be developed at the XLib level.

1. Use of XIM and XOM

XIM is the international input method protocol defined by X. With this support, people can input characters into most of the X applications. All the input methods are modularised and can be dynamically loaded in most of the supported platforms.

In an internationalized program, any particular mapping between keystrokes and input characters cannot be assumed. This program must run in any locale on a single machine. In X, still there has been no concrete solution provided for this problem. The following problems were faced, while trying to implement the XIM/XOM:

1. There is no documentation or clear idea regarding the implementation of XIM/XOM.

2. Input method is still glyph based. It is necessary to implement it in encoding based.

For Indian Languages, simple XIM/XOM is insufficient. XomCTL (X Output Method with Complex Text Layout) Support is also regarded. XomCTL is still under development [XIM].

2. Modification at X-library Level

The modifications are made to the XLib so that, any X application inherits the changes, irrespective of the toolkits. Applications need not be recompiled.

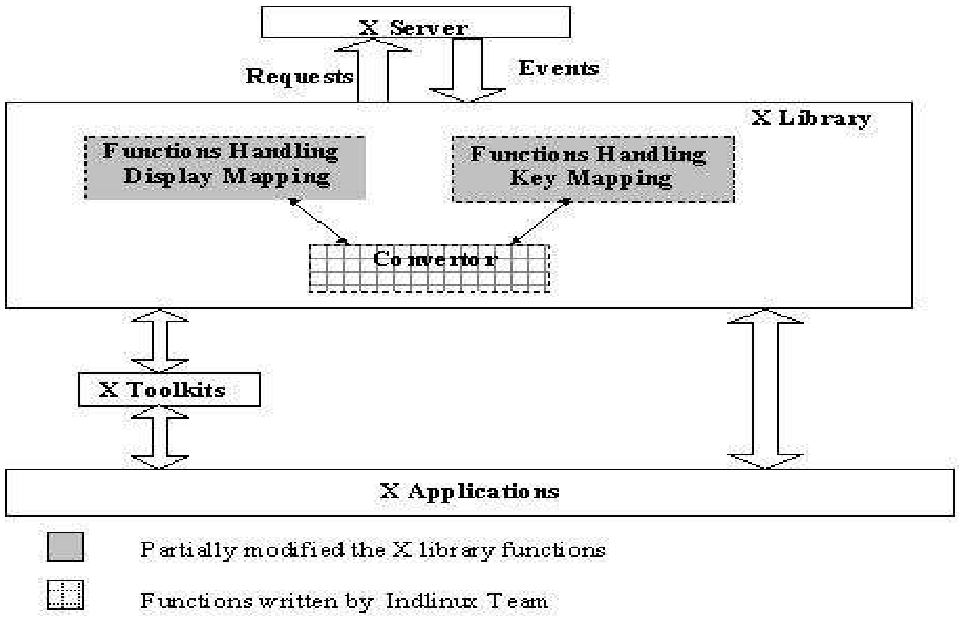
X Server
Requests
Events
X Library
Functions Handling Display Mapping
Functions Handling Key Mapping
Convertor
X Toolkits
X Applications
Partially modified the X library functions
Functions written by Indlinux Team

Figure 1: Architecture of modified X-Window System

The basic difference between supporting ASCII and Indian Language is that Indian Languages are made up of character clusters, which are produced by a sequence of keystrokes. Further the glyph(s) displayed can also depend on multiple key sequences and are context sensitive. This requires that the sequence of keystrokes be remembered. To enable this, a virtual buffer is used to remember the past keystrokes.

A few variables have been added to the display structure and a convertor for converting ASCII to encoding code and encoding code to glyph indices are provided. These are explained in the following sections. Figure 1 explains the modified architecture of the X.

The solution is provided in two different ways:

a) Encoding Based

In this method, certain encoding standards are used to store the file contents. The file can be opened and edited in any operating system or applications, which supports this encoding.

b) Glyph Based

In this method, the input is converted to glyph. File contents are stored in the form of glyphs. It is easy to implement but the drawback is that it depends on fonts.

4.2.1 Keyboard Mechanism

In localizing X, the major issue is, mapping keycodes generated by keyboard to native character codes and displaying the native characters. This issue is dealt as follows:

1. **Encoding Based**

   In this module, input code(s) are converted to encoding code(s). In the Xlib function, XkbLookupKeyBinding(), a variable 'Indian_Mode' is defined. When a toggle key is pressed, (to switch between English and native language) the value of 'Indian_Mode' variable is changed between true and false.

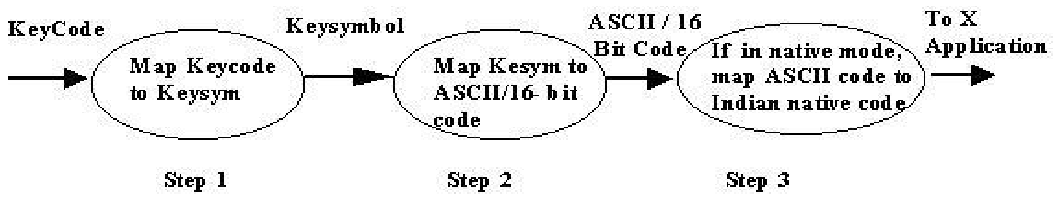

Figure 2: Keyboard Mechanism

In XkbTranslateKeySym function, conversion is done as follows:

```
If ( Indian_Mode ) then
If ( KeyMap table is not loaded ) then
Load the KeyMap table
Endif
Output_KeySym = Ascii2Encode( Input_KeySym )
Else
Output_KeySym = Input_KeySym
Endif
Send the Output_KeySym to the application.
```

Since there are many language-scripts in India, we need to make a generalized code to represent all the scripts. As a result an encoding code, which represents native code, is sent from Step 3.

2. Glyph Based

Here the first two steps are similar to the encoding mechanism. In the third step, whenever X is in Indian_Mode, the input key events flow through the Key Event Filter and the resulting events generated by the filter are sent to the application. This process is handled as follows:

```
If ( Indian_Mode ) then
Convert the Keycode to glyphs
Delete_Glyph = length( Previous_Glyphs )
Cluster_Output_Glyph = EventFilter ( Previous_Glyphs, Current_Glyphs )
If ( Cluster_Output_Glyph != Current_Glyphs ) then
Generate Delete_Glyph numbers of backspace events
Endif
Generate Key Events to send the Cluster_Output_Glyph to Application
Endif
```

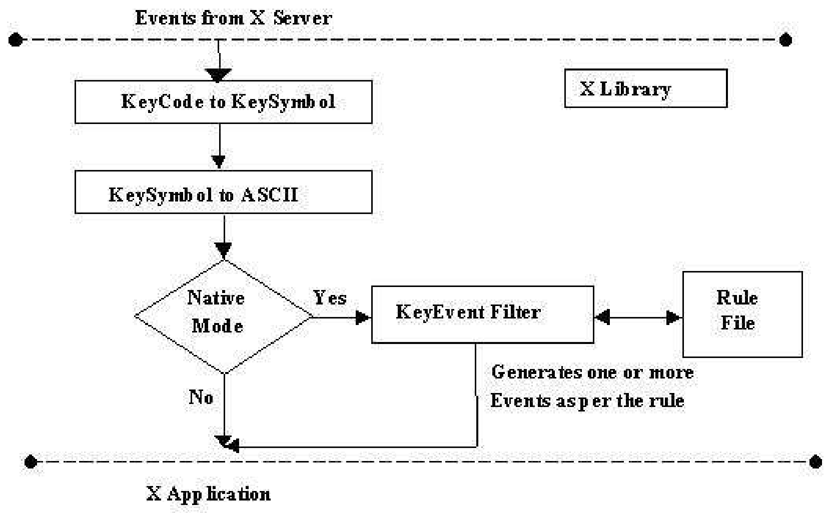

Figure 3: Glyph based Keyboard Mechanism

Key event handler has the information about, keycode, keysymbol etc. In the Key Event Filter, the Keysymbol(ASCII) is checked with the rule file and it finds the matching glyphs. The new Key Events are generated as per the result from the rule file and the keysymbols are replaced by the glyphs.

Since input characters are converted to glyphs, there is no need to handle the display process. Similar to the encoding based approach, a toggle key is used to switch between English and Indian language.

2. Display Mechanism
1. Encoding Based

In order to incorporate Indian languages support to X, some of the XLib functions such as XDrawImageString, XDrawImageString16, XDrawString, XDrawString16 are modified.

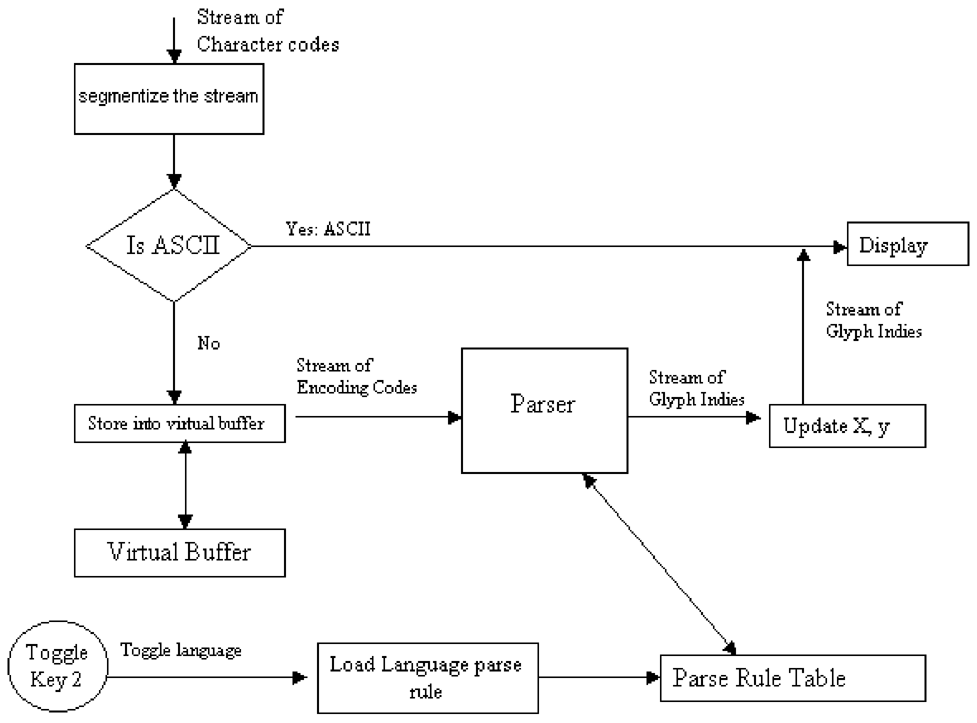

Figure 4: Display Mechanism

The following steps have been checked to these 8-bit draw functions:

1. Segment the incoming stream of character codes.
2. Check each incoming code and break into groups of subsequent ASCII codes and native codes.
3. If it is the group of ASCII codes, send it for display.
4. Else, get the stream of glyph indices corresponding to the native codes using a function match_parse_rule() and send it for display on the screen.
5. Update the (x, y) coordinates based on the stream of glyphs.
6. Repeat step 3, 4 and 5 for all the groups.

For XDrawString16() and XDrawImageString16() functions, each of the character code is represented by the structure Xchar2b. In these functions, if it is 7-bit ASCII or 8-bit native code, the higher order byte will be zero. In this case, the codes have been treated as 8 -bit. Otherwise, the appropriate language has to be identified from the range of the code and the corresponding parse rule is loaded.

In X, the display mechanism maintains no history. In order to handle the cursor-positioning problem, which is a major issue, the history is maintained about the current cursor position.

1. Glyph Based

The input side handles conversion of input code to glyph code and clustering. If an event is generated, the input mechanism checks the key symbol and does the following:

1. If no cluster formation, it forwards the event to the application.

2. If cluster occurs, it generates the events for erasing previous glyphs and sends the event for displaying current cluster.

1. Editing

Editing mainly includes operation of deletion, backspacing and cursor movements. The application handles all the editing operations

1. Encoding Based

The editing in the encoding based is a very trivial task. The application knows only the stored encoded content. It sends the encoded code to XLib for display purpose. The XLib converts them to glyphs and displays it. Since the application is not aware of this conversion, the cursor location, decided by the application, may not match with the display. Since, cursor is not in an appropriate position, deletion and backspace are issues.

Handling cursor position in case of Indian languages is complicated. When a vowel modifier follows a consonant, the clustered character must replace the consonant and cursor position must be adjusted. The same behavior must be there in case of the deletion or backspacing also. Cursor positioning plays a crucial role for applications handling variable width font, especially, in the case of the Indian characters, the cursor must be moved forward or backward by the width of the present or previous character respectively.

2. Glyph Based

The application handles all the editing operations and no modification is done in display mechanism. The cursor is therefore positioned properly. Most Indian characters are made up of a number of glyphs. To delete a character, we need press backspace/delete key a number of times.

2. Convertor and Rule file

1. **Encoding Based**

   The convertor is a collection of functions and tables. It follows the encoding mechanism for converting the keycodes to meaningful native characters. This has two parts. One is for converting ASCII to Encoding Code (KeyMap) and the other, for converting encoding Code to Glyph Code (DisplayMap). Each encoding mechanism has two rule files. For the KeyMap part, a keylayout table is used, which is language independent and for the DisplayMap part, a map table is used (language dependent).

2. **Glyph Based**

Only one keymap file is used for one language. A keymap file is enough for the entire process.

3. Automata

The parse rules are used to construct deterministic finite automata (DFA). The automaton helps in determining the cluster formation and related information. As an illustration, DFA for a set of four parse rules are given in Figure 5.

Parse rule matching is used while characters are being displayed. Suppose that there is a parse rule α β A = a b c. Also assumes that the glyphs α β are already displayed. Now, when the character code corresponding to A is pressed, the DFA matching is initialized. If the rules are already loaded before, and then the traversal will go in the order A -> β -> α , matching all successive characters, finally when it encounters a non-matching character, the data element in the last DFA node (abc) is returned.

α β A	➔	a b c
β A	➔	d c
γ A	➔	e c
α β B	➔	a d c

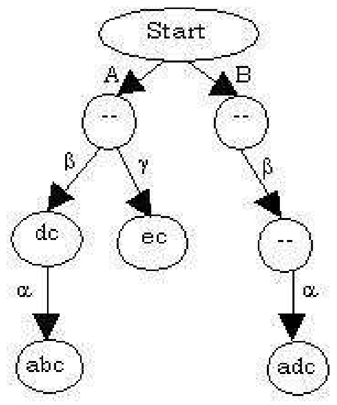

Figure 5: Finite State Automata

In the encoding based solution, Display map file is used to construct the DFA. This file is used while displaying the characters. In the glyph based solution, Language rule file is used to construct the DFA. This file is used to convert the input keySymbol to glyphs.

1. Applications

There are few applications, which has been localized using the tool GNU gettext [GNU]. List of localized applications include: Compiler (gcc), Mail client (pine, Kmail, Netscape), Shell (Bash), Browser (lynx, Netscape, Konqueror) and Office Suite (KOffice, Database, and OpenOffice).

2. Conclusion

The modifications made in the X Library provide the support for any Indian language or any language that demands variable width font and/or uses modifiers and clusters. The task of providing support for any Indian language, thus reduced to, creation of bi-lingual font and creation of the parse-rules as appropriates for the language.

We hope that this work will go a long way in the context of NLS for software and consequently, in the process of taking the benefits of "information revolution" to the marginalized sections of society and to achieve appropriate social use of IT.

References

[Kenistion, 1998]Kenneth Kenistion, "*Politics, Culture and Software*" Economic and Political Weekly, Vol. cXXXIII, No, 3, pp. 105-110, January 17, 1998

[ISCII] *Indian Standard, Indian Script Code for Information Interchange - ISCII* from Bureau of Indian Standards, bisind@del2.vsnl.net.in

[XIM] The Free Standard Group Linux Internationalization Initiative, URL: http://www.openi18n.org/subgroups/im/

[GNU] GNU gettext package documentation, available at the URL: www.gnu.org/software/gettext/gettext.html