8. Text Processing in IndiX

Sandeep Rao, Staff Scientist, CDAC, Vinod Kumar, Software Specialist, CDAC
E-mail : {sandeep, vinod}@ncst.ernet.in

I. INTRODUCT ION

GNU/Linux was to be localized so that *most applications can work almost as easily with Indic scripts as with English*. With this goal in mind, this project named IndiX, identified the necessary and minimal changes to existing infrastructure and carried out these changes to support Indic text on GNU/Linux platform. The development was based on standards, especially Unicode. Progress was made to define new standards, for example for the visual syllable of characters. Lastly existing deficient standards were addressed and improved eg. Indic OpenType standards [5].

Developing a local language capability at system level is better than developing it at an applica-tion level. The former enables all the applications running on the top of the system to exercise that capability. Thus the approach that IndiX followed was to *make changes to the software at the core level so that widest range of applications benefit*. This can be seen from Fig. 1. The more lower you make the infrastructural changes, the more wider is the range of impact. Hence in IndiX, we have modified the lowermost layer of the X Windowing system, the XServer, to support Indic scripts [5].

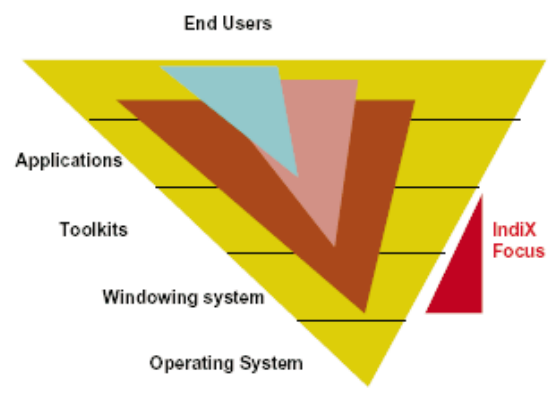

Fig. 1 The maximum impact philosophy

Current software, shaping and font architecture for handling multilingual text has evolved from the ASCII and Latin centric viewpoint. While the representation of text using the Unicode standard, especially with UTF-8, has received widespread acceptance, the software interfaces for editing and rendering text have still not achieved uniformity or standardization. It is not enough to translate a message to a new language and place it in a locale of the language. Interfaces to render such messages are the least that are necessary. Any application that interacts with the user would also require interfaces for inputting, and editing as well. As editing and rendering text is the core functionality to be supported by multi-lingual software, we in-vestigate how the ASCII and Latin centric software interfaces for rendering can be upgraded to support complext text of Unicode characters.

II. TEXT RENDERING

The most basic capability needed in text pro-cessing is the one to convert a logical sequence of characters to a renderable sequence of glyphs.

A. Rendering Latin text

The unit of orthogrphy in Latin scripts is the character. The CharMap in a font for Latin script implements the one-to-one mapping from character to glyph.

Fig. 2 shows that X11 handles the character to glyphs transformation in the X11 Server. The X11 Server is closest to the the rasterizing pipeline and the display hardware. The Latin text is passed as characters to the X11 Server through a narrow interface.

B. Rendering Indic text

The unit of orthogrphy in Indic scripts is the syllable. An Indic syllable consists of one or more characters that can be rendered by a sequence of glyphs. The

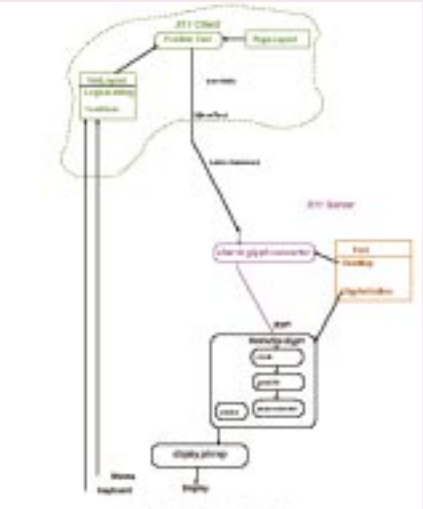

Fig. 2 XII rendering Latin text

interactions between characters will be confined to be within the syllable. A character within a syllable will not move to or change the shape of the preceding or succeeding syllable. Thus Indic characters map to glyphs. This is the most general mapping from characters to glyphs advocated in ISO/IEC TR 15285 [1].

The object model of the ISO/IEC TR 15285 is shown in Fig. 3. A Syllable can be Logical syl-lable or a Renderable syllable. Further, a Logical syllable can be an ordered sequence of Character, or an ordered sequence of Glyph.A Renderable on the other hand is an ordered sequence of Glyph.

An Intelligent Font that has additional informa-tion on how a sequence of coded characters is transformed into a sequence of glyph identifiers, with associated position information is needed to implement the many-to-many CharsToGlyph and Glyphs ToGlyphs relations. These transformations are carried out by a pipeline. The pipeline will have to be presented with nothing short of a syllable.

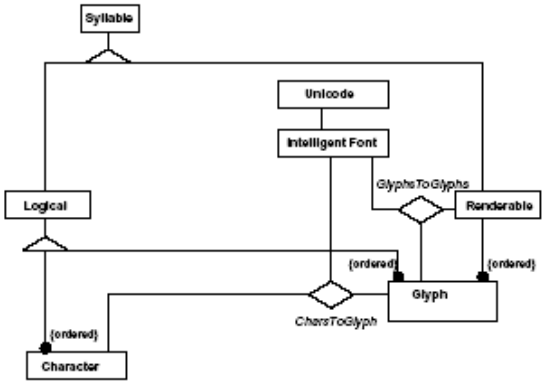

Fig. 3. Object Model for characters and ghyphs. ISO TP 15285

Fig. 4 shows the Indic shaping pipeline inte-grated into the X11 Server. OpenType fonts im-plement the Intelligent font.

III. EDITING OPERATIONS

The enhancement to the X11 Server at the bot-tom should have enabled Indic text for all higher levels. But editing is a higher level operation that needs further models and related operations.

A. Editing in Latin script

Visual editing using the mouse to position the cursor and then inserting a character by using the keyboard or deleting a character is facilitated by using a TextLayout object. The TextLayout con-tains the logical character string, and the extents ("lengths") of the glyphs corresponding to each character. It need not have the glyphs directly but only their extents. It is important to distinguish be-tween the character offset and the physical position on the display. The TextLayout object will have methods that will convert from a hit position (x, y given by mouse) to the character offset within the logical string. In the other direction, TextLayout can give the position to place a caret if you give it the offset of the logical character after which the caret should appear.

Consider a string Lazy dog jmped over. A char-acter u is to be inserted after j. The insertion operation is carried out with the following steps:

1) The user places the cursor using the mouse after j, clicks and types u.

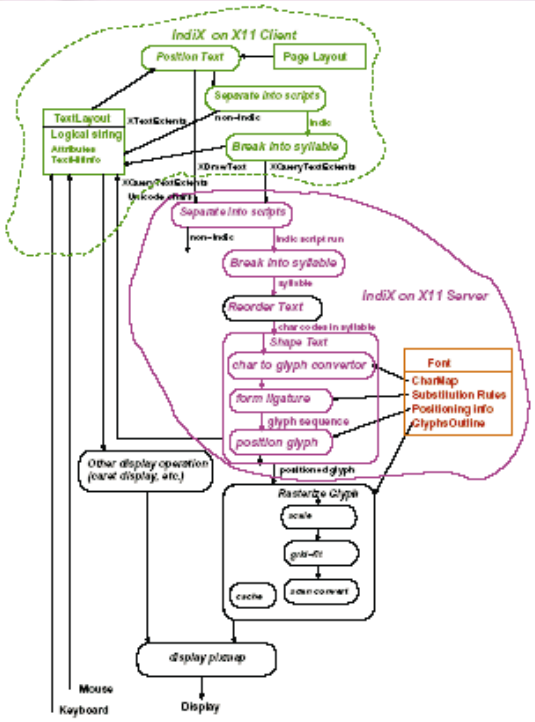

Fig. 4 XII rendering Indic text

2) The coordinates of the cursor are intimated to the editor program.

3) Using TextLayout, the editor maps the po-sition to be after character j.

4) The editor adds u between j and m in the logical character string in the Textlayout.

5) The editor clears the display till the end of line after the position of j.

6) The editor redisplays the string umped over. from the TextLayout at starting position ob-tained from the Text Layout for the character u.

B. Editing in Indic scripts

In Indic scripts, the syllable is the orthographic unit equivalent to a character. If the editing oper-ations delete or insert in unit of a syllable, then the Latin framework for editing would work for Indic scripts provided we replace character with syllable. But we cannot wish away the fact that the syllable is made up of one or more characters. The extent of a syllable is not the sum of the extents of the glyphs of its component characters. More over, the editing operations in Indic scripts are built up from insertion or deletion of characters and not of syllables. So we study the effect of characters and character level editing on the syllables.

1) The TextLayout contents: The TextLayout object having Devanagari text of two syllables is shown in Fig. 5. Fig. 6. shows, in the last line, the two syllables rendered on a display. Coming back to Fig. 5. the line labeled Logical syllables of characters shows the five logically ordered Uni-code characters of the text. The sequence 0930 094d 0915 is the first syllable. And the sequence 0930 093f the second. The boundaries of the two are shown in the line labeled Boundaries of Logical syllables. The line labeled Caret offset shows the position between the characters where a conceptual point of editing has been selected by the user. The use of this attribute will be described later. The syllable boundaries and the caret offset are expressed in terms of the character offset and not in terms of physical position on the display. For relating the syllable boundaries to the physical position, we need the syllable extents. The next line labeled Extents of Renderable syllables shows how much space each syllable takes on the display. The first syllable takes 1856 FontUnits and so does the second.

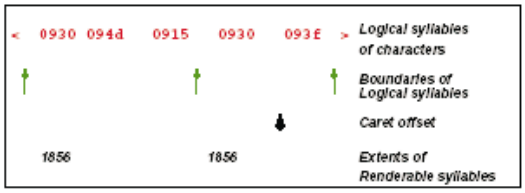

Fig. 5. The TextLayout object

2) How are the TextLayout contents made?: The logical syllables of characters are obtained from the keyboard input or a text stream. When the five characters are in, the editor program submits the sequence to a syllable boundary extractor. This interface scans the logical characters, and detects syllable boundaries according to the Unicode stan-dards [3]. Once the syllable boundaries are identi-fied, the editor submits these syllables to another interface that returns the extent of each syllable.

A syllable's extent is available only after passing the syllable through a shaping pipeline for Indic scripts. It is illustrated in Fig. 6. The logical sylla-ble of characters is fed into the shaping pipeline. Out come the sequence of renderable syllables. The interface routines sum the advance widths of the renderable glyphs of a syllable to obtain the extent of that syllable. So the first syllable has an extent of 1856 (= 1856 +0). The second syllable too has an extent of 1856 (=744+1112). This is just a coincidence and not because it is a mono spaced font!

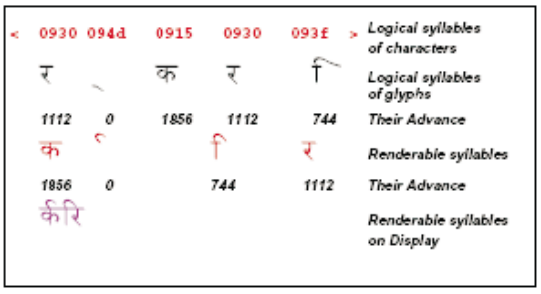

Fig. 6. Making of the two syllables

If the syllable has to be displayed then the renderable glyphs are submitted to the rasterization and display pipelines. When the extents alone are required, this step can be avoided.

The line labeled Logical syllables of glyphs (Fig. 6) shows the glyphs corresponding to each character. These glyphs are obtained by using the CharMap in the font. The advances of these glyphs are shown in the next line. Summing the advances of the glyphs in the Logical syllable of glyphs will not give the correct extent of each syllable. In Latin, many editors and toolkits find the extent by using the CharMap. It will fail for Indic scripts. The extent for Indic text can be computed only after getting the final renderable glyphs.

In IndiX, the syllable boundary extractor is available both at the X11 Client as well as the X11 Server. The TextLayout, being at the client, uses the one at the X11 Client. The invoker does not have to separate the characters into different script ranges. Since the Shape Text pipeline is in the XServer, the extent of a syllable has to be obtained by a query XQueryTextExtents that is answered by the XServer (Fig. 4).. The extent is dependent on the script of the syllable and the font used for the script, but the XServer determines the script and chooses the font allocated to the script. This allocation is generally by default but the client can change it by a SelectFont method. Thus neither operation, syllable extraction nor extent evaluation is script dependent for the editor application.

3) Editing behaviour: The Unicode Standard Annexe # 29 [3] describes how editing opera-tions like mouse selection, arrow key movement, backspacing have to behave uniformly in a syllabic environment. The Annexe identifies the grapheme cluster (equivalent to renderable syllable of glyphs) as the unit over which these operations take place. In some systems, the delete key would delete the previous grapheme cluster where as the back space would delete a character of that subcluster.

Directional arrow may navigate over the charac-ters. But displaying the position over the compo-nent characters is problematic. In IndiX, the caret is shown only between two syllables. The user has to mentally count the number of back arrows typed in and try back space. If the count was correct, the unwanted character would be deleted and the new syllable (or syllables) would be displayed. Alternative would be to have a tool tip window showing the individual glyphs of the logically ordered syllable with the caret in position. Instead of a tool tip, the syllable into which a caret has moved in could be displayed in the logical order in line.

4) Character level editing on syllables: Contin-uing with the example (Fig. 5), the user inserts a Halant (094d) character between the two syllable. The six characters now form a single syllable (Fig. 7). So the effect of insertion of a character is deletion of two syllables and the insertion of a new syllable! There could be various such cases depending on whether the operation is a character insertion or deletion, where it is being inserted (at beginning, middle or end of a syllable) or what character is being inserted or deleted. To reduce the complexity, and make the editing operations independent of scripts, IndiX suggests the follow-ing procedure to handle character level edits.

   1) If the editing point has been selected by a mouse click, then identify the syllable to which it relates and fix the caret offset in the TextLayout at the character associated with the nearest syllable boundary. Place a caret on the Display at the physical position associated with the boundary.

   2) If there is an arrow controlled movement then move the caret offset in the TextLayout over a character. Do not redisplay the physi-cal caret unless the caret offset crosses over to the adjacent syllable.

   3) After positioning, if the user inserts or deletes a character, identify left and right edit-buck-stops-here syllable boundaries. The syllables beyond these left and right boundaries will not be affected by the edit operation. The characters between these boundaries may regroup into new syllables. If the caret is not at a syllable boundary in the TextLayout but

somewhere in between a syllable, then edit-buck-stops-here bound-aries are those of that syllable. If the caret is positioned at a syllable boundary, then the left edit-buck-stops-here boundary is the left boundary of the previous syllable. The right edit-buck-stops-here boundary is the right boundary of the next syllable.

4) Dissolve any syllable boundary between the left and right edit-buck-stops-here bound-aries, and clear the extents. The characters between the edit-buck-stops-here boundaries are not associated with any syllable after the dissolution and are free. After a edit opera-tion, at most two syllables will be dissolved.

5) In the freed character sequence, insert or delete the edited character.

6) Update the caret offset in the TextLayout.

7) Resubmit the new character sequence to the syllable boundary extractor and associate the new boundaries with the new character se-quence in the TextLayout.

8) Evaluate the extents of the newly created syllables and update the TextLayout.

9) On the display, clear the line starting at the physical position associated with the left edit-buck-stops-here boundary.

10) Redisplay the characters after the left edit-buck-stops-here boundary at the same phys-ical position.

In the example shown in Fig. 7, these steps are applied as follows:

1) The user clicks the mouse somewhere in between the two syllables (Fig. 5). A click event at, say 1860 (in FontUnits) is intimated to the editor application. There are three boundaries in this TextLayout, first one be-fore the first character (0930), the second after the third character (0915) and the third after the last character (093f). From the ex-tents of the syllables, these boundaries have the physical position at at 0, 1856, and 3712 (=1856+1856). The click at 1860 is clearly closest to the second boundary (1856). The boundary is between the characters 0915 and 0930. The Caret offset in the TextLayout is made to point between these characters. The XServer is asked to display a caret at 1856, the physical coordinate of the second boundary. Note that the click was at 1860 but the caret appears at 1856.

2) The user keys in the 094d (Halant) character. The event is intimated to the editor applica-tion.

3) The application starts looking for the edit-buck-stops-here boundaries in the TextLayout. It sees that the caret is after third char-acter in the TextLayout and it is the second syllable boundary. So the two syllables to the left and right will be affected by the insertion. The application determines that the left boundary of the previous syllable, the one before the first character (0930), is the left edit-buck-stops-here boundary. The right one is the after the fifth character (093f). All these are from the TextLayout object and the application does not look at or ask anything from the shaping pipeline or the display.

4) The application dissolves the second bound-ary (after character 0915) that is between these two edit-buck-stops-here ones. The two extents 1856 and 1856 are erased.

5) The application inserts the character (094d) in the Logical syllables of characters (Fig. 7).

6) The caret offset is moved to point between the characters 094d and 0930.

7) The new sequence of six characters is sub-mitted to the syllable boundary extractor. The extractors returns saying that the se-quence is a single syllable and that there is no syllable boundary within the sequence. The boundaries in the TextLayout are up- dated as shown in Fig. 7.

Fig. 7. Insertion of a character leads to deletion of 2 syllables and insertion of new syllable

Fig. 8. Making of the single syllable

8) The single syllable of characters is submitted to the extent evaluation interface. The mak-ing of the single syllable is shown in Fig. 8. The extent (2600 FontUnits) is added to the TextLayout.

9) The line on the display is cleared from position 0 onwards.

10) The syllables of characters after the left edit-buck-stops-here boundary is submitted for display starting at physical position 0 (FontUnits).

The implementation of the Text Layout opera-tions on IndiX follows naturally from the IndiX and X11 architecture. We identify the syllable boundaries in the Logical string in the TextLayout. To get the extents of the syllables, we invoke XQueryTextExtents and store the extent of each syllable in the Attributes structure.

Most of the other X11 based Indic text handling environments [4], [2] locate the Shape Text pipeline on the X11 Client, probably to simplify the Text-Layout operations.

C. Summary of enhancements for Indic editing

1) An interface to extract the syllable bound-aries from a logical sequence of characters is invoked first. This interface should be script independent. Any script dependency should be handled transparently by the implementa-tion of the interface.

2) Second interface returns the extent of a syl-lable. The interface provided to toolkits and applications should be script and font inde-pendent. But there should be default fonts for each script. Toolkits should have facility to change the default font.

3) A TextLayout object, that stores the logical syllable of characters, the syllable bound-aries, syllable extents and the caret offset facilitates the editing operations. The object should have methods to convert between physical position on the display to character and syllable boundaries.

4) Character level editing is first handled by dissolution of old syllables and creation of new syllables. The display of the modified text is handled after the new syllables have been generated in a fashion similar to that of Latin for new characters.

5) The TextLayout and the editor should sup-port a command to recalculate the TextLayout attributes. This is needed when the font for a script has been changed. Automatic recalculation is complex as the higher level interfaces for the TextLayout calculations are independent of font or font selection.

IV. CONCLUSI ON

If the software has to handle multilingual text, the interfaces for the text editing and rendering operations should be internationalized. The starting point of this process can be the Latin script editing and rendering architecture. The changes from this have to be minimal, necessary and logical. Layer-ing of functionality has to be adhered to. Editing interfaces should not redo the work done by lower layers.

The two most important changes from the Latin architecture that we have identified in the IndiX project are for the character to glyphs transforma-tion and the text editing.

V. REFERENCES

[1] Edwin Hart and Alan Griffee. An operational model for characters and glyphs. Technical Report ISO/IEC TR 15285, 1998.

[2] IBM and Others. International Components for Unicode (ICU) ICU 2.8 Documentation. http://oss.software.ibm.com/icu/apiref/, 2003.

[3] Mark Davis. Unicode Standard Annex # 29 Text Boundaries. Technical report, The Unicode Consortium, http://www.unicode.org/reports/tr29/tr29-4.html, March 2004.

[4] Owen Taylor. Design Goals, and Reference documentation for Pango. http://www.pango.org/index.shtml, 2003.

[5] Rekha Sharma, Ajit Joshi, Sandeep Rao, Vinod Kumar. Unicode for Multilingual Software: An Indian Perspective. In 25th Internation-alization and Unicode Conference (IUC25), Washington, DC, Mar-Apr 2004.