Unified Text Layout Engine For FLOSS Systems

Vinod Kumar &Siji Sunny, C-DAC Mumbai

At present, the Textual User Interface programs of FLOSS-based desktop systems often use different text layout engines: many GTK-based programs, including The Gimp and Inkscape, use Pango as the text layout engine; Openâ€ Office uses IBM's ICU text layout classes; KDE programs use Qt's layout engine; while many applications use homegrown text layout engines: KOffice uses Qt's native shaping but has its own code for paragraph layout; and Scribus currently has its own code for the whole layout process but the development team is thinking about using a forked version of Qt's shaper. Because of differences in the layout engines operating â€œbehind the scenesâ€ , different software can exhibit differing levels of support for complex text layout (CTL) scripts like Arabic or Kannada. A unified text layout engine,independent of any specific toolkit,would allow developers to focus first on rendering text in all modern human scripts correctly and secondly , on doing so efficiently. (almost verbatim copy from http://live.gnome.org/UnifiedTextLayoutEngine)

The idea of **Unified Text Layout Engine For FLOSS** was discussed at Open Discussion held at *Gnome Live 2006* Boston™. The topic generated a lot of interest in FLOSS developers, but the outcome is not available as a recommendation. So as a continuation to that meet, we can carry forward the standardization of Unified Text Layout Engine in this conference and probe towards a constructive outcome.

The existing text layout engines differ not only architecturally but also in the nitty gritty of their interfaces. It is impossible to unify the interfaces before a consensus on the architectural framework is reached. Moreover, once the architecture is in place, individual interfaces can be wrapped in Adaptors to present the unified interfaces. One of the observations the discussion at Gnome Live 2006 was: *Existing APIs of pango, Qt and probably ICU must be maintained: otherwise it won't be a unified text layout engine, just yet another text layout engine*. The Unified Text layout architecture and the adaptors for the existing toolkits appear to be the only way out.

Indix Approach

IndiX has strong opinions on the architecture and interfaces of a Unified Text Layout engine. IndiX has backed up its architecture with a successful implementation. We would like to articulate and present the IndiX view at a Unified Text Layout Engine in FOSS.IN 2007, Mozilla project day. We hope that the interaction and contribution will result in an Architecture for the Unified Text Layout Engine and later for its interfaces. Here we outline some of the architectural and interface issues where IndiX subscribes to or differs from others.

- IndiX is of the opinion that the architecture of ICU is the best of the lot, conforming to the Unicode text processing principles. ICU is not an also ran. Unicode and ICU generalize the *One code One Glyph* of Latin to *Many Codes Many Glyphs* style of complex scripts. Moreover, ICU realizes the importance of treating *Many Codes* as a unit in layers where the corresponding *Glyph*s do not appear. Thus ICU has a specialized `Character`

`Boundary` Iterator to return the *Many Codes* in a text sequence. These interfaces are useful in editing operations such as highlighting, selection, cut and paste that inherently work on the text in the back up store. Such operations do not require the glyphs but only their extents. ICU handles the *Character to Glyphs* transformations through LayoutEngines that encapsulate the details of the fonts and transformation within specialized `OpenType Layout` or `AAT Layout` objects. ICU architecture should be the basis of the Unified Text Layout.

- IndiX seeks to identify text processing components at all layers of the software architecture and enhance that component and its interfaces logically, and minimally so that the software works for all Unicode scripts, not only English. Hence the architecture we suggest for Unified Text Layout should address not only the script related issues like Character boundaries but also issues relate to the layered architecture of software. An example would make this clear. With the X11 Graphical User interface used in Linux/Unix systems, the X11 Client layer handles the character sequences and the X11 Server carries out the characters to glyphs transformation and the display of the glyphs. When the X11 architecture is used with the ICU library, several 'crossed transactions' (as in Transactional analysis in Games People Play by Eric Berne http://en.wikipedia.org/wiki/Transactional_analysis) occur. At the upper X11 client layer, the lower layer ICU Layout engine is exercised to get the glyphs just to know the extent of those glyphs- a crossed transaction. Now assume a X11 Server that has implemented the Characters to Glyphs transformation using ICU. (We know of no such X11 Server. Only IndiX has done this but it is based on the superset of the ICU architecture). This lower layer will have to break the text string at the syllable (the multi-character sequence that is displayed a one shape) boundaries using the upper layer `Character Boundary` Iterator before converting them into glyphs using the font- another crossed transaction. Let us indicate how IndiX reduces or avoids such crossed transactions. In the X11 Client, the text is marked up with syllable boundaries. To get the extent of the syllables, the lower layer X11 Server is called but only for the extents. The X11 Server does not return the glyphs although it does have to carry out a full *Characters to Glyphs* transformation and sum of the extent of each glyph to obtain the text extent. *Prima facie* there is a crossed transaction here as bad as in the non-Indix case. When the text has to be displayed, the marked up sequence is sent to the X11 Server. In IndiX, the X11 Server can extract the syllables one by one and convert each into its glyphs, all in a completely script independent fashion.
- IndiX has based the grouping and reordering of characters on the Unicode standard and the minimal model of fonts. IndiX uses the ISO/IEC Technical Report 15285, which gives an operational model for transforming characters to glyphs. The most general transformation from M characters to N glyphs and the intelligent font recommended by the Technical Report 15285 have to be used with IndiX scripts. IndiX has suggested two enhancements to the model. The first is to treat certain character sequences, mainly for the consonant signs, as an indivisible sub cluster. The second is a visual order for characters different from the logical order of characters. So in IndiX, shaping of Indic text from characters to glyphs is done at the lower most font level. Currently OpenType is the Intelligent font technology used for this. IndiX clearly separates

page layout operations from the OpenType layout operations. The OpenType operations are for converting from M characters to N glyphs without the use of any feature flag like HALF. IndiX is of the considered and proven idea that OpenType features are controls only to modify the normal transformation of a character to a glyph. They need not be mis-used to shape complex scripts like South Asian ones. A half Ka and full Pa glyphs should and can be generated from <Ka Halant Pa> without any further specification. Creating "Indic OT Specifications" that require Open Type Features like HALF, setting these features on the characters <Ka Halant> and tagging the OpenType font tables for half consonants with HALF, all lead to complicated software and fonts, problems with interoperability and difficulty in designing, understanding and using the fonts. It is clear who benefits from such round about ways of laying out text- not the Free and Open Source community nor the user. There is a tendency to treat Complex Text Layout and Open Type Layout as one. This thesis has to be smashed from two fronts. First of all Complex Text Layout can be solved with other Intelligent font technologies (see IEC 15285 and AAT Layout in ICU) not only OpenType. Secondly, OpenType is being misused and made to look like if its features (pun intended) are what makes Complex Text Layout possible.