Title: Computer support for Signwriting written form of sign language

Author: Guylhem Aznar <lrec@externe.net>, Patrice Dalle <dalle@irit.fr> TCI team, IRIT lab <www.irit.fr>

Keywords: Sign language, Computer, writing, Signwriting, Unicode, Linux

Abstract: Signwriting's thesaurus is very large. It consists of 425 basic symbols, split in 60 groups from 10 categories. Each basic symbol can have 4 different representations, 6 different fillings and 16 different spatial rotations.

While signwriting is more and more used by the deaf community, it currently lacks a complete and platform neutral computer support to let signwriters share documents regardless the applications and the underlying operating system they may be using. Based on previous research, various propositions have been made, resulting in multiple incompatible systems.

The main problem currently is the lack of a consistent basis upon which compatibility could be built : the most advanced and used system, SWML [1], is multiplatform thanks to Java but requires dedicated applications like the previous attempts. Moreover, the use of XML based representation requires dozens of lines of code for each symbol, resulting in oversized files which can not be parsed, used or read with standard tools. XML linking to bitmap pictures for on-screen representation prevents the integration of a real font system, needed for a true portability, and cause scalability problems. Moreover, like previous systems, SWML still comes with a complex user interface, a little easier to learn but slower, symbols being entered via the mouse.

Even if this advanced approach helped the signwriter community, replacing the manual insertion of GIF graphic files for each symbol, at the moment, the signwriting community must revert to screenshots and pictures to ensure documents can be shared and read, resulting in little reusability for both users and researchers, and low computational possibilities worsened by the absence of signwriting optical recognition software.

Guylhem Aznar, a first year medical resident and a PhD student in Computer Science from Pr. Patrice Dalle TCI team in IRIT (Toulouse, France), is proposing a unicode based representation for Signwriting with a suite of free software tools running on GNU/Linux but also supporting non-free operating systems.

This approach based on unicode is putting a strong emphasis on facilitating communication and compatibility through a unicode reconstruction engine. Usage and computer entry are also made simpler thanks to different possibilities of human interaction : keyboard, mouse and sensitive area (handwriting) support, which all result in the same unicode-text output. This output can then be shared, reused or studied easily.

The choice of unicode over XML facilitates integration in existing software.

The system works in layers : the entry layer, the keycode layer, the unicode layer, the rendering layer and the font layer.

These layers are independent and therefore easy to adapt and improve.

In the keycode layer, each signwriting **“basic symbol”** is coded by a different number called *“internal name”*. This basic symbol is first positionned geometrically by **“positionning elements”** defining concentric circles and the respective angular position of the basic symbol on these circles. The basic symbols can be completed by additional information regarding the possible variations, such as spatial rotations, required in order to form the complete symbol. These **“additional information elements”**, like the basic symbols and the positionning elements, are also coded by one or more numbers also called internal names. All these internal names are linked to their respective meanings in a mapping table. Additional internal names can be defined following the evolution of signwriting's standard. Finally, **“delimitors”** are used to group basic symbols into complete signwriting units.

In the unicode layer, another mapping table is used : these internal names are mapped to unique unicode characters. One or more internal name can be mapped to a unicode character, but each unicode character can only have one mapping. This non-bijective approach is required to follow the unicode standard.

In the entry layer, signwriting symbols can be entered by different peripherals like a keyboard or a mouse. The mouse driven graphical input system will be completed by other entry modes in the future. Following the traditional key mapping entry mode, a table maps internal names to the physical keys on the keyboard. Multiple keyboard mapping tables allow different physical dispositions for different countries or following user preferences.

The entry layer is separated from the rest of the system. It is only relevant to the system by its dependancy on the unicode layer, required in order to output unicode characters following the keycode layer specifications.

In the rendering layer, a unicode reconstruction engine like Gnome's Pango, transform the flow of unicode characters into a graphical representation, i.e. a complete signwriting symbol. It is not yet suitable to the display: elements are still numbers (then called *“external names”*), and must be replaced by graphics. The transformation is coded by a set of rules [3] describing the possible combination and the outputs, like for unicode arabic and indian languages support.

In the font layer, a font subsystem like Gnome's Freetype/xft2, which support both traditional bitmap fonts and vectorial fonts, takes care of the graphical representation, replacing external names by their corresponding graphical symbols.

Different fonts can of course be used.

Considering a symbol has been entered though the entry layer, it must then be transcribed into a serie of unicode characters following these steps:
- first, a **positionning element** is used to define a circle. If this circle preceded by another circle before the initial **delimitor**, it is embedded in that circle. A special type of circle is used to define the contour of the face
- then, **basic symbols** are positionned on that circle, with **positionning elements** to define their angular position followed by **additional information elements** if these basic symbols need rotations, special fillings, etc.
- finally, a **delimitor** is used to mark the end of the signwriting unit.

The internal names of these entities are never used – instead, unicode characters are used, which allows existing software to process signwriting. These unicode caracters are then mapped to the internal names, and the rendering layer geometrically and spatially reconstruct a complete signwriting unit in the form of external names. The font layer then replaces this information by the graphical drawing of the complete unit.

Currently, the different layers are under work. They do not require the same amout of work: the most complicated part is the definition of rules for the rendering layer [4], the hardest task is drawing fonts, the most important is the keycode layer to provide a quick replacement to SWML and the longest part is reserving enough space in unicode for a correct signwriting implementation. The latter may eventually be impossible, in which case “private” unicodes areas will have to be used. This should only cause some minor changes in the unicode layer, but will damage the portability benefits of using unicode.

This entire “text-like” layered approach makes a clear separation between the various sub-systems used, providing a solid base upon which new sub-systems can be built (for ex. in the entry layer, handwriting recognition) and any layer can be upgraded (ex: adding additional vectorial fonts, supporting a new signwriting standard) without requiring a full system redesign. Applications following Gnome's API can immediately take advantage of signwriting support, which means a whole desktop suite of software is made available for free to deaf-users. Moreover, signwriting features (ex: writing from top to bottom) no longer need special handling through specific applications, thanks to Gnome localisation support.

An additional advantage is the portability of the model. Support on the GNU/Linux based PDAs requires no further work. Windows or MacOS support would require minimal support in the entry layer and at some specific points in the font layer.

The upcoming support of Windows and MacOS by Gnome applications means these steps could also simply be removed in the short term. Moreover, Signwriting transcription in standardized unicode text means the text can be subject to automated computer analysis, exchanged by researchers, etc.

Possible evolutions of the system include a statistical approach for auto completion and handwriting recognition, and will certainly focus on the user interface with the design of specific Gnome Accessibility features.

References:

[1] Rosenberg, A. Writing Signed Language, “In Support of Adopting an ASL Writing System”, Dept. of Linguistics, Univ. of Kansas, USA, 1999
http://www.signwriting.org/forums/research/rese010.html

[2] Antonio Carlos da Rocha Costa and Gracaliz Pereira Dimuro, “A SignWriting-Based Approach to Sign Language Processing", Universidade Catholica de Pelotas, Brasil, 2001
http://www.techfak.uni-bielefeld.de/ags/wbski/gw2001book/draftpapers/gw38.pdf

[3] Klaus Lagally, “ArabTe{X} : typesetting arabic with vowels and ligatures”, 1992
http://citeseer.nj.nec.com/rd/64325440%2C95192%2C1%2C0.25%2CDownload/http://citeseer.nj.nec.com/cache/papers/cs/350/ftp:zSzzSzftp.informatik.uni-stuttgart.dezSzpubzSzlibraryzSzncstrl.ustuttgart_fizSzTR-1992-07zSzTR-1992-07.pdf/lagally92arabtex.pdf

[4] Finite State Automata and Arabic Writing - Michel Fanton Certal-Inalco
http://acl.ldc.upenn.edu/W/W98/W98-1004.pdf