Elements of Computational Ethiopics

ዳንኤል ያዕቆብ መኰንን
Daniel Yacob, *dmulholl@cs.indiana.edu*

June 1997

Abstract

Ethiopic software development has journeyed through a maturity process that has advanced now to some adolescent stage. Further maturation however has in some cases been stymied by limitations of the computer environments themselves. But this is becoming less so the case and Ethiopic support in popular operating systems is poised to jump through the final steps to adulthood. This particular period of time is set to become the most interesting and promising since Ethiopic first made its way onto computers.

Facing the end of traditional limitations, developers, and to some extent the software users, will need to become more familiar with some new concepts that are encountered in multilingual environments. In this paper an overview of each will be made with an emphasis on the orthogonality of the abstractions each area represents and their relevance to Ethiopic computing.

Ethiopic computerisation as a starting point

With the exceptions perhaps of the occasional video game or scientific analysis we are using computers to work with text documents. The Internet, for all its over-hyped glory, is more than anything else a conduit for us to exchange text in such rewarding forms as email, web documents, and interactive chats. Of course we must let our less adept friends believe we are doing more than just that. The communication between us and the machine is again through text. Files have names using text, menus use text, dialogs and messages ("Ok", "Cancel", "Exit") use text, mice and icons provide shortcuts to reduce the tedium of working with so much text. Alas, the keyboard still reigns supreme.

Herein lies the nature of our problem; <verb> the machines to mimic our understanding of text. For us "A" is simply "A" and "ሀ" is simply "ሀ". Each the first letter in their writing systems but usually we do not need to know any more than this, their names and sounds, to understand "A" and "ሀ". Lacking a brain, computers will never have our subtle understanding, or any understanding, but we can convert "A" and "ሀ" into numbers for them and they will get by.

Computers will treat all characters (the smallest element of text) as numbers. But not always the same number, it may depend on what we are doing with the character. Typing in the character through the keyboard "ሀ" can be understood as one or many code numbers -depending on the typing system. Displaying "ሀ" on the screen, a different purpose, "ሀ" may be understood as a completely different number -which can depend on the font used to display "ሀ". Saving the file to the disk, E-mailing, or transferring between computers in different ways, "ሀ" and its 400 or so odd siblings may dress in the clothing of still different numbers.

All of these number systems are what the computer needs to pretend to understand "ሀ" or any letter. If we work with text in these different ways and "ሀ" at some point comes out as "ዠ" or anything else, the computer failed to understand "ሀ" and we have not solved our problem.

When all of the Ethiopic characters can safely be represented as numbers in these different areas

where we use text, when the association between the numbers and letter is robust within the environment; we have "computerised Ethiopic". But to safely type, save, send and receive Ethiopic text is only the start of what a document composer might wish to do with the text. When the characters have been computerised we can then use them to build words and words need sorting, searching, conflating, hyphenating, spell checking, and sometimes complex manipulations in data bases. When computers and software can use computerised Ethiopic to communicate to us in words -in the menus and dialogs, careful considerations are needed.

There is more to Ethiopic computing than simply the computerisation of Ethiopic. This author feels "Ethiopic Computerisation" is an over simplification of the field. Certainly not all computerisation issues are yet satisfactorily resolved but resolutions are at hand. Close enough that the industry is already moving into an era where algorithms and applications are being developed to address the problems unique the Ethiopian user's needs. Similarly, internationalised and localisable software allow interface to be configured for the Ethiopic user's needs and expectations. Research is also underway that addresses text and information processing unique for Ethiopic script and languages.

Collectively we can think of all areas as "Computational Ethiopics". <expand> Lets look now at the areas that

Character encoding

In the human-computer relationship is it at the monitor or at the keyboard that man and machine are first united? Or is it there we are divided? It is primarily through the keyboard that we will communicate with the computer and (aside from warning beeps) through the monitor that the computer communicates to us. Two devices fusing us in a silent symbiotic partnership. Lets start with the monitor, or rather with the understanding of the written word on the other side.

We see in a luminant glass plate a flow of text giving us the familiar shapes of paragraphs, borders, pages. In the paragraph, text breaks into sentences composed of phrases and ultimately the flow is made up of letters that when arranged properly, give form to the familiar and unthought of shape of words. The elements of meaning. For our partner the same is just a block of memory where every eighth bit discerns a number. Meaning can only be in the numbers now. But what numbers and why?

In ASCII "A" is 65, in EBCDIC "A" is 193. This example is the encoding of the character "A". "A" has a different encoded value in these two systems. Through character encoding we are requesting our partner to understand text, stored perhaps in a file or in memory, in a given way (either as 65 or 193). Hopefully our partner obliges and presents the text to us with an appropriate font that has the same encoding, or understanding, of letters. Encoding then is what truly unites machine with man in the world behind the glass.

Assigning the character codes (numbers) to Ethiopic text elements is half of the job in Ethiopic computerisation. Each player in the field has elected to use a different encoding system (like ASCII vs EBCDIC in our example) which complicates the exchange of text as we will discuss next.

Information interchange

Information interchange may be thought of as assuring the successful communication or transfer of a computer's understanding of text to other computers. We are usually not confronted with having to deal with our partner's understanding of text until the text is misassociated (misunderstood) with a wrong encoding system. In the far east, software users are confronted with a variety of encoding standards and have developed some savvy for how to deal with them. Most software for the far east languages will support multiple encoding standards to cope with this reality.

The same reality is prevalent in communication with Ethiopic script. When a document is saved on a disk as a file, or transmitted around the world as an email message, the encoding system travels

as an intrisic part of the text. Will the person opening the file later as an email, web page, or word processor document see the text number 65 as "ሀ", "A", "ዣ", or garbage? In the case of the email and web page MIME provides a mechanism to transfer the document text encoding with the document.

The computers on both sides may have been computerised for Ethiopic independently but must share the *same* understanding of Ethiopic text on both ends to exchange information successfully. Usually, understanding of an encoding system will correspond directly to having the proper Ethiopic font installed. When this is not the case, a computational Ethiopics application could provide multiple understandings by way of translation between encoding systems. So that the received text could then be presented correctly on the monitor for the user at the other end.

Also under the area of information interchange a developer may need to be aware of byte order and how many bits are assured to travel safely across a medium. But these and related issues of transfer and their solutions are the problems of exchanging bits, independent from the encoding super imposed upon the bits. Hence secondary encodings of 16 or 8 bit systems for 7 bit transfer is not an issue unique to computational Ethiopics so we do not explore the matter here. Transliteration of Ethiopic text may be applied for information interchange but serves other purposes as well. We will discuss transliteration as a subject in itself shortly, lets look back to how man talks to machine.

Input method

Input method (or IM) is what forms the other half of our relationship with the computer, it is how we communicate the text we want to our partner. Pressing down a key communicates one number, a key code, to the computer. Releasing the key communicates a second number. One or both may then be translated into the number of a letter in our character encoding system. We may compose a character with multiple keystrokes, perhaps adding a "Shift", "Meta", or "Alt" before our previous example to produce another letter.

To the Ethiopic software user the input method may be the software's defining feature for ease of use. Is "ሀ" to be composed with '`h + e`', '`h + v`', or only '`h`'? The preferred system is probably the one we are already used to and not necessarily the most logical by some other measure. Consider how the legacy of the Ethiopian typewriter lives on in the the software world. For linguistic reasons, in particular the slightly different phoneme and writing conventions of the different language groups using Ethiopic script, users may also insist on a diversity of Ethiopic IMs.

Lets be clear though that key codes do not correspond directly to character codes. In the above example pressing down the "`h`" key would signal to the computer the scan code of 35, which corresponds to the key code of 35 in a "pressed" state. Releasing the key the computer receives the signal for the scan code of 163, corresponding to the key code of 35 in the "released" state. Upon pressing down the key, and sending the key code signal 35 to our partner, we were probably presented with character code 104 on our screen. Character code 104 could have either the familiar shape of "`h`" or "ሀ" depending on our current font's character encoding. In the second event, the release of the '`h`' key, likely nothing new happened on our screen as our partner was informed of the key's release.

If we wished to have both Roman, Ethiopic, and other scripts in a single font (reducing the burden of changing fonts as we type) the key code 35 could still correspond to character code 104 when we wished to type in Roman script *and* key code 35 could correspond to character code of 4608 for "ሀ" as we type in Ethiopic script[1]. We can break away from a one-to-one correspondence between key codes and character codes when we have an intelligent IM interpreter that we can inform of our current context (perhaps through a menu or a meta key or sequence).

There need not be any meaningful correspondence between the key codes and character codes. In the case of ideographic writing systems (Chinese,

[1] This is exactly the case in a Unicoded encoded font.

Japanese, Korean) popular IM systems have users spell out the names or sounds of the ideographic characters which are looked up in a "dictionary" and possible choices are presented in a menu for selection.

By no means must IM be through a computer keyboard either, it is simply the most prevalent, the natural extension of the typewriter. Lets consider the extreme case of a computer without a keyboard. The mouse alone might suffice to enter text, perhaps selected from a palette one character at a time, or one word at a time. Either way this is too tedious for us. Keyboardless IM is no small matter when you consider the handicapped user who may be without one or both limbs. Special one handed (left or right) keyboards are available as well as speech recognition systems that turn spoken words into commands or typed text. More advanced systems aid the less fortunate still and provide IM by monitoring eye movement.

Implementations of IM abound, often they are a separate process from the application receiving the focus of the text. IM may run as an "Engine" or "Editor" (IME) serving a single application at a time on a single computer or be run as a server over a network (IIIM or Wnn) serving many applications and computers simultaneously.

Transliteration preprocessing

There is an area in Ethiopic computing that tends to be applied in this three areas that we have just explored (character encoding, information interchange and input method). Understandably then it is often confused to be one in the same with one or more of these areas though it is really something quite different and despite the secondary applications actually has a purpose all its own. Lets now take a careful look at transliteration.

Transliteration

Transliteration is an area important to computational Ethiopics as the need for it arises for different computing purposes. The objective of transliteration is to represent the characters of an alphabetical or syllabic system of writing by the characters of a conversion alphabet. As there are a gamut of target alphabets that users will want to transliterate into (or retransliterate from), as there are multiple phoneme conventions for the Ethiopic syllabary, and any number of different computing restrictions, a small variety of transliteration conventions have surfaced.

Each transliteration system is designed to serve the same purpose, they differ primarily from their initial premisi. We need not review each existing system and its design criterea or the context for which each would be applied. Our purpose here is only to establish an understanding for transliteration and an appreciate of it as different in purpose from other areas in computational Ethiopics.

Transliteration of Ethiopic text is a process that is performed when the text could not otherwise be preserved. <How does this relate to understanding?> The reason for why the text could not have been preserved (we assume electronically preserved) will govern the design of the transliteration system itself. Our wish and purpose is only this, we will likely restore the text later either cognitively or electronically.

It is *not* necessary that the transliterion onto the target writing system preserve the spoken level of the original text. Though it is normal to do so. When our purpose is to preserve and communicate phonemic values in the target system more so than the original text elements, our process becomes that of transcription.

<give an example from personal names, mamo and michael, note the difference and that transliteration is easier to preserve>

A keyboard input method can also provide a transliteration convention for a writing system; likewise a transliteration system may also be applied as an input method. When the transliteration process is generally performed by hand it is argueably convenient to unite the two. The objectives of IM and transliteration are not however at the same time unified. The efficiency of each have different measures and over relating the two will not produce an optimal system for either.

As we established before a character encoding

system could also provide an IM system (the usual case with ASCII key mappings) though we do not necessarily always want or have to use it. Similarly a character encoding system could also provide a transliteration convention. This is even applied for Ethiopic script regularly when the writing system was mapped onto ANSI addresses and saved as text. Here character encoding and transliteration is a little clouded in part because transliteration is assumed to apply to human scripts and not computer character sets.

Where a non-Roman writing system mapped onto only the alphabetic portion of the computer character set we can argue that it is also produces a transliteration system. This was case for Greek and Cyrllic practices in early computers.

Again, while possible to unite the transliteration, character encoding, and even IM system, it is for the majority of the world's languages not advantageous to do so (except in Latin when considering it as a trivial transliteration onto itself). The same issues of efficiency and differences in purpose apply.

Transliteration as we have said serves different purposes and may take different forms as conditions mandate. Transilteration is important in computational Ethiopics because it gives a way to work with Ethiopic text when otherwise we would not be able to (primarily in ASCII only environments). Transliteration may be applied for information interchange under 6 and 7 bit restrictions, for file names, variable or macro names (which may be case insensitive), and to compose Ethiopic documents when it can not be done natively.

The developer may have to apply a transliteration or be prepared to process a text stream in transliteration. ...so we should have standards...

Text & Information Processing

Like transliteration and transcription, text and information processing are generally used interchangeably or at best text processing might be argued as a type of information processing. Be that as it may, I would like to make some distinctions that I find useful though you may not find them applied in such a way elsewhere.

Text processing are the events that happen at a logical level where text elements are read by a software, identified, understood, and manipulated into different understandings as would be necessary in a task. Text processing is often the conversion of text between different encoding systems for the purpose of information interchange. For instance Ethiopic text may need conversion into a 7 bit system for email -possibly into SERA, MIME, or UTF-7. Or perhaps into or from UTF-8 to write or read an Ethiopic world wide web page. Compiling the Ethiopic macros in a LaTeX document, searching, sorting and interpreting conversion of key codes into character codes are also examples of text processing.

Text processing work can also go beyond the level of a single character and identify and process words blindly for such purposes as word counting, selection, and formatting needs like justification, indentation, line breaking, or punctuation rules evaluation. Optical character recognition, or when Multilingual Emacs (Mule) editor offers features to modify only the form of an Ethiopic syllable and exchange ASCII and Ethiopic word spaces in a region of text. These are again text processing areas where we manipulate text with only a concern for its numeric character code value.

When we need to become more concerned for the surface level of the text, or its value; work upon and manipulation of the surface level or value of the text becomes information processing. When LaTeX compiles an Ethiopic document and applies the rules of Ethiopic hyphenation to formatting, or when a text processor modifies terms in a religious article to the color red, we are in the area of information processing. When Mule offers to reduce Ethiopic numerals to one of three different levels for transliteration, this is information processing.

Information processing is by far the more challenging, at the very high levels it becomes computational linguistics. Information processing for computational linguistics is what must be done in the analysis of a word to evaluate spelling or grammar. Intelligent searching whereby a part of

speech could be conflated or stemmed under the rules of Ethiopic orthography is highly important in Ethiopic information processing. Developing efficient methods for these are currently research areas. A language translator from Ge'ez to Tigrigna, or German would also be computational linguistics as well as information processing.

Localisation

Another area where the contextual level of words in languages becomes important in programming is at the user interface. We return to the monitor now and how the software will communicate to us. As much as the colour and cosmetic attributes of the software the logical arrangement of menus and the application vocabulary define the ease of use and "feel" of a software.

The menu and message vocabulary, and their proper formatting, is now at the level of importance in computer applications, including the operating system itself, that the issue has rightly become a field of its own. Applications made for the international market will more likely than not apply National Language Support (NLS) as well as other interface customisations through locales.

Under the NLS approach application messages are not written explicitly in computer source code. Rather, they are maintained in an external file, a seperate file for each language, and referred to at run time by a reference name or number. The application may determine the default language by querying the operating system or a user preference file. The language of the application UI might even be selectable from a menu.

The use of external messages, called "catalogues", is only one aspect of "localisation" (or "L10N" for short)[2] which should be understood as the adaption of software to the needs and expectations of users in a given market. The market boundaries generally follow the language groups within a country and distinctions are made for British vs American, Australian and Canadian English as well as other internationally spoken languages. Specific settings to a language and writing practice within a region is then called its "locale". These preferences are then also kept seperate from the application in locale file stored in some common locale directory on the system.

When a language setting is changed for an application or the computing environment as a whole there are other important localisations that must be consider. A locale for a given language and cultural convention will also specify formats sort order, keyboard layout, font sets, date, time, number and currency formats. It may come as a surprise to some programmers that the C character class testing and mapping functions, such as `isalpha`, `isspace`, `tolower`, etc., will return different results depending on the current locale setting.

While not going into detail in this summary, hopefully the possibilities of L10N have been sufficiently highlighted. The main consequence for software that has been localised through use of locales is that the software will likely be able to support languages for which the designers knew nothing about at compile time. When we realise then that most operating systems and many applications since the mid 90's have been able to support locales a new burden is placed upon the Ethiopics developer community: To specify and provide locales for Ethiopic languages. The specification and design of locales is for the most part fairly simple. However, such details in an Ethiopian locale as the calendar system and the use of two interword spaces in text will likely call for implementation at the algorithmic level.

Onward: Computational Ethiopics

I have avoided giving a recommendation in these different areas as my purpose in this article is only to illustrate the different areas, and their distinctions, that compose the areas of computational Ethiopics. These areas discussed in brief here appear to be the most relavent to the needs of

[2] The abbreviation to "L10N" may not be obvious at first: Localisation = L + ocalisatio + n = L + 10 letters + n = L10N.

Ethiopic computing and to the users in the present day. Unquestionably the field will grow along with the computer industry and computational Ethiopics will always be those collections of subsets that require specialised reasoning for Ethiopic script, languages, and cultural conventions.

A final area in wait to be addressed by developers in computational Ethiopics is the establishment of some central entity of focus in the field. Establishing such a body in the field would give industry a conduit for communication to propose and establish standards, to provided computing and information resources for multilingual software developers, and a representative body for communicating with the major software companies and world standards organizations.

This task before the industry, academics and government and private science organisations, may well be the most challenging to tackle in our new field.