Internationalizing XTF

Bringing Multilingual Features to the Extensible Text Framework

Martin Haye
Email: martin@snyder-haye.com
May 2005

INTRODUCTION

CDL deals with a great deal of text, and not all of it is in English. Despite being based on technologies that support Unicode (Java, XML, Lucene, and Saxon) XTF is currently English-centric, with limited support for Western European relatives such as French, German, Spanish, and Italian. But the world is much more than Europe and North America, and many interesting difficulties will be encountered in the process of supporting texts from the Middle East, Asia, and elsewhere.

"Full" support in XTF for all the world's languages is an almost limitless task. For practical purposes, internationalizing XTF can be viewed as a progression:

1. Fill out support for searching Western European language text in mostly English documents. In this same phase, it should be possible to support other languages that use simple alphabetic scripts, such as Greek, Cyrillic, and derivatives.
2. Add facilities for searching non-European languages written with Latin script, and for documents in European languages containing no English text at all, but still using an English-only user interface.
3. Support searching texts written in selected non-Latin scripts, such as Arabic, Hebrew, Devanagari, Chinese, Japanese, and Korean.
4. Support non-English localized user interfaces, so that a French, Russian, or Chinese "skin" could be selected by the user.

In the following sections, anticipated problems in each of these steps will be discussed.

The non-XTF information in this paper is largely based on (and sometimes lifted directly from) the excellent book *Unicode Demystified* by Richard Gillam. Those interested in exploring humanity's fascinating variety of written languages – and how they are represented within computers – are encouraged to pick up this book.

In addition, CDL's Mike McKenna made extensive comments and provided valuable information which has been incorporated into this paper. His input is very much appreciated.

1. WESTERN EUROPEAN LANGUAGES

Many languages originating in Europe share Latin and Greek as common ancestors. Now spoken around the world, these languages also share a common alphabet and writing system (with some variations of course.)

Currently, XTF contains limited support for searching non-English language text written with the Latin alphabet. Essentially, the user can search for words containing characters from the ISO-8851 ("Latin-1") code page. This covers words from French, German, Spanish, Italian, the Scandinavian languages, and a few others.

However, someone trying to use XTF to search for non-English words would encounter many minor irritants, and so the first step for XTF is to make the user experience better in these cases.

Most of the transformations in the sections below should be performed on documents when indexed and on queries when they are received from a user.

1.1 Unicode Composite Sequences

In Unicode there may be several ways to represent what a user would consider to be the same "character." This is due mainly to the direct incorporation of all the various national and international standards, to speed adoption. For instance, the character **ö** can be represented two ways:

U+00F6 LATIN SMALL LETTER O WITH DIAERESIS

or

U+006F LATIN SMALL LETTER O, *followed by*
U+0308 COMBINING DIAERESIS

Currently a document indexed and displayed in XTF might use either of these representations of **ö**, and a user search for one wouldn't find the other. For this Unicode problem, Unicode provides an answer by defining various "normalized" forms. The most common one is called Normalized Form C and is the standard defined by the W3C for processing documents on the World Wide Web. In our example, the two code point form of **ö** would always map to the single code point U+00F6.

Another problem is letters with multiple diacritical marks, such an **o** with a circumflex above and a dot below. Unicode allows the two marks to be stored in either order, but normalization defines a fixed ordering, allowing proper searching and comparison.

It has been suggested that Unicode "Sort Keys" be used to address these problems. See section 2.5 for a discussion of the benefits and downsides of sort keys.

Recommendation: Convert all input documents and user queries to Normalized Form C, also known as NFC. See http://www.unicode.org/reports/tr15/

1.2 Queries in HTTP

Unfortunately, the syntax for URLs on the World Wide Web is very restrictive: it allows only ASCII Latin letters and digits, plus a handful of special symbols and punctuation characters.

While this allows them to work across many systems and protocols, it obviously presents a problem for transmitting many non-ASCII characters.

Note that this is a different problem than Internationalized Domain Names (IDN), which XTF should be able to handle now without any difficulty. For a discussion on IDN, see *http://www.icann.org/topics/idn.html*.

There is a general convergence in the industry on encoding URL parameters in UTF8 and then translating to the "%" syntax. However, this is far from universal, and we may encounter many browsers that do entirely different things. Currently XTF uses the HTTP **get** method and hence inherits this problem if users try to search for non-ASCII characters.

If this turns into a problem, perhaps the best way to address it is to make the initial search page into an HTML **<form>** element and use the HTTP **post** method. There is a much more standard way of marking and transmitting arbitrary characters in a **post** transaction. Subsequent links that don't involve the user entering text can still safely generate and use URLs and HTTP **get**, because XTF can enforce the UTF8 encoding.

According to Mike McKenna, HTTP **post** is the most reliable way to send UTF8 data. However, he points out that this problem may eventually resolve itself for us. Users are steadily upgrading to new browsers, reducing the proportion of older problematic browsers. Eventually those problem browsers will form too small a fraction to worry about.

Recommendation: Try waiting out the problem, but if necessary, add support for HTTP **post** as well as **get**.

1.3 Alternate Character Representations

Several languages (even English) allow more than one way to represent the same character in the user's mind, even if it isn't the same display glyph. Here are some examples:

	var 1	var 2	example
German	**ß**	**ss**	bi**ß**en -> bi**ss**en
	ö	**oe**	m**ö**gen -> m**oe**gen
English	**ö**	**o**	co**ö**perate -> co**o**perate
	ï	**i**	na**ï**ve -> na**i**ve
	æ	**ae**	arch**æ**ology -> arch**ae**ology
	é	**e**	r**é**sum**é** -> r**e**sum**e**
Obsolete English	**ſ**	**s**	pre**ſ**ident -> president
			congre**ſ**s -> congress
Icelandic / Norwegian, Faroese	**æ**	**ae**	Æðelwald -> AEðelwald
	ð	**th**	*(It is very rare for* **ð** *and* **þ**
	þ	**th**	*and to be transliterated.)*

As you can see from the example of **ö**, this mapping isn't the same for every language. In German it properly maps to **oe**, in English the accent is ignorable. The simplest answer is to err on the side of more generalized finding, by enabling XTF's ability to ignore different

diacritical marks. This may annoy users when they find too much, but at least the results will contain what they're looking for.

Another area of slight concern is that of case folding. The Greek letter sigma has two lowercase versions, one that occurs at the end of a word and another that occurs in the middle of a word. Currently the XTF indexer treats them separately, since it maps all strings to lowercase. Cases such as Greek sigma and many others are taken care of by simply mapping first to uppercase, then mapping the result to lowercase. There are a few rare cases that are not handled by this (notably the Turkish dotted I vs. dotless I), but they can be safely ignored until we have localized UIs.

It's been suggested that Unicode "Sort Keys" provide a solution to address accent insensitivity. See section 2.5 below for a discussion.

Recommendation: Enable accent insensitivity by default. Add additional mappings for **ß**, **æ**, **ſ**, and the like.

1.4 Punctuation, Numbers and Symbols

For compatibility reasons, Unicode includes many "presentation" forms of characters. For instance, there are codes for subscripted versions of 0 through 9, as well as superscripted, circled, parenthesized, and with a period. These should probably be mapped to the normal digit set, so that, for instance, one could search for **E=MC2** and still match **E=MC²**.

Unicode contains fraction characters for **¼**, **½**, etc. These should be mapped to **1/4**, **1/2**, etc.

Additionally, the copyright symbol **©** should be mapped to **(C)**, and the trademark symbol **™** should be mapped to **(TM)**. There are other instances of this sort of mapping.

There are various hyphen characters in Unicode that should for the most part be ignored by the indexer (currently they split up words in the indexer.) In particular, the "soft hyphen" should not be considered to break a word, and in fact should be removed from any word when indexed.

There are at least two ways to represent a negative number: using an ASCII "hyphen-minus sign" sign, and with the Unicode "minus sign." These should both be mapped, probably to the ASCII.

There are some special mathematical symbols which may be used in documents but a typical user is unlikely to type. There are several invisible symbols which should be ignored by the indexer, such as U+2061 INVISIBLE TIMES. There are several operators such as U+2215 DIVISION SLASH that should be mapped to more-common ASCII characters. Then there are many special forms of letters used in math formulas, such as U+2113 SCRIPT SMALL L, that should be mapped to their ASCII equivalents.

Various countries punctuate numbers differently. People from America, England, and France would punctuate the same number three different ways:

1,234.56 vs. 1.234,56 vs 1 234,56

It might not be obvious that 3,14159 and 3.14159 are both the beginning of the number *pi*. In addition, there is no universal standard of where to put separators. In India one might write "40,10,20,300.45 rupees" instead of "401,020,300.45 rupees" as a westerner might.

To circumvent confusion and broaden search hits, XTF should probably attempt to remove numeric punctuation from documents.

Unicode contains specific glyphs representing, for instance, "Roman numeral 8," and these should be mapped to multi-character sequences such as **VIII** and the like. Additionally, early Roman numerals were written with characters not in the Latin alphabet. For instance, 1,000 was written with a character looking like a **D** with a mirror image **D** attached to the left. Some scholarly works may contain this character, and it should probably be mapped to the modern usage, **M**. This data is all available in the Unicode character database.

Again, Unicode "Sort Keys" were suggested as a possible solution; see section 2.5 below.

Recommendation: At index and query time, map alternate digit, letter, and Roman numeral presentation forms, plus © and ™, to their ASCII equivalents. Map "minus" and other math operators to ASCII. Disregard invisible math operators and soft hyphens. Remove numeric punctuation when possible.

1.5 Unicode version

In the past, we've claimed that XTF supports Unicode but never said which version. It turns out there are many versions of Unicode, and it looks better if we specify which one we support. XTF will inherit most of its Unicode support from Java, so we simply need to advertise the Unicode version of the latest Java runtime we support.

2. OTHER LANGUAGES USING LATIN SCRIPT

The "Latin" alphabet is ubiquitous and used for many entirely unrelated languages the world over. The following sections will discuss a few additional issues concerning two main areas: first, indexing documents that are entirely in Western European non-English languages, and second, indexing those written partially or completely in other languages but with Latin script.

2.1 Stop Words and Plural Forms

XTF has special processing for very common words, called "stop words." Here are some stop words:

English: a an and the of I me he she it is we they them her his…
French: a et la le l' de du moi il elle est…
German: ein einen und der die das den dem ich er sie es sein…

Because they're so common, queries containing stop words are normally time-consuming to process, since each occurrence of the stop word must be examined. XTF melds stop words to adjacent words to form bi-grams and thus speeds query processing substantially.

There are occasional cases where the same word is a stop word in one language and not in another. Consider **the** in English vs. **thé** in French (the tepid drink, "tea"), or **die** in English vs. **die** in German (definite article, "the"). However, since in XTF stop words aren't third-class citizens and act almost like normal words, it's probably safe to ignore these few cases of cross-language overlap.

So to deal with additional languages, we need only add stop words from each language to the list used by the indexer. Many of these stop-word lists can be found in the European "Language Engineering" community.

XTF also has a facility to perform "plural mapping", where a user could query for **cat** and find either **cat** or **cats**. This mapping is strictly dictionary-based, so adding plurals from all the languages in question would be most expedient. There may be an issue with variant pluralization of words depending on the gender of the context, but these variants should be loadable from dictionary files as well. Also, there might be a slight problem with cross-language overlap, but these are rare and the potential impact is slight.

Recommendation: For all languages where we plan to index significant quantities of text, add stop words and plural forms.

2.2 Special Sounds

Some characters are used to represent special sounds not normally written in English; unfortunately, most users don't know how to type these characters and so mapping them is desirable.

For instance, when writing "Hawai'i" the apostrophe actually represents a sound – in this case a glottal stop – and Unicode thus assigns this usage a different code: U+02BC MODIFIER LETTER APOSTROPHE. However, most users will simply type an apostrophe. Similarly, some African orthography uses **!** to represent a "click" sound, but most users will type an exclamation point. The special Unicode characters should be mapped at index time to their more common counterparts.

Unicode "Sort Keys" have been suggested as a solution to this, but probably don't help in this case. See section 2.6 below for a discussion of sort keys.

Recommendation: Add mappings for special letters to their ASCII equivalents.

2.3 Transliteration

Serbo-Croatian is written using both the Latin and Cyrillic alphabets: the Serbs and Montenegrins use the Cyrillic alphabet, and the Croats and Bosnian Muslims use the Latin alphabet. Serbo-Croatian spelling is standardized so that there's a simple one-to-one mapping between the letters in the Serbian (i.e., Cyrillic) alphabet and the Croatian (i.e., Latin) alphabet.

A user might rightly expect to search for a word using Latin letters and find words written in Serbian, or vice-versa. To support this, the XTF text indexer would have to support some form of "transliteration", indexing each word in both of its forms so it can be searched either way.

One could go farther down the transliteration road, for example mapping words in the entire Cyrillic alphabet to letters on an English keyboard, and similarly for the Greek alphabet. One can put together a fairly robust regular-expression based transliteration engine. In fact, the International Components for Unicode (ICU) contains a good engine that is also free. See http://www-306.ibm.com/software/globalization/icu.

However, it may be difficult to get satisfactory transliterations. ICU does a reasonable job, allowing one to choose between *transcriptions*, which are source-target language based, and *transliterations*, which are reversible glyph-for-glyph transforms. This deserves some experimentation.

Recommendation: Experiment with the transcriptions and transliterations available in ICU, and incorporate any that produce satisfactory results.

2.4 Grapheme Clusters

XTF supports wildcard matching. Entering **?** matches one character, and entering ***** matches any number of characters. Unfortunately, "character" in the previous sentence refers to a Java character, that is, a 16-bit unit. This isn't necessarily what the user thinks of as a "character"; hence the unwieldy but useful term "grapheme cluster".

Consider this glyph: **õ̱**

In the memory of the computer, this is represented by three "characters":

U+006F LATIN SMALL LETTER O
U+0303 COMBINING TILDE
U+0323 COMBINING DOT BELOW

If the text contains the term **dõ̱g**, the user should be able to find it by entering **d?g** into the search box.

Things get worse: Unicode defines a large coding space, with 2^{21} possible code points. There are a large number of (often rare) characters that don't fit into the 16 binary bits Java allocates to a character. In these cases, a "surrogate pair" of two characters is allocated to a single Unicode code point. One certainly wouldn't want the **?** wildcard to match half of a pair but not the other half.

The Java system library apparently supports determining grapheme cluster boundaries.

It has been suggested that Unicode "Sort Keys" may automatically get rid of troubles with grapheme clusters; see section 2.5 below.

Recommendation: Test and incorporate Java's grapheme cluster boundary detection into the XTF text indexer and query engine.

2.5 Unicode Sort Keys

Unicode Technical Report #10 defines the Unicode Collation Algorithm (UCA); see http://www.unicode.org/unicode/reports/tr10/. This is a clever method of sorting Unicode strings in a locale-sensitive manner, and specifically deals with of many of the problems with alternate encodings discussed above, along with accent and case sensitivity.

A full discussion of the algorithm is beyond the scope of this paper, but in essence, it transforms each input string to a "sort key". The first (or "primary") part of the sort key is more or less a copy of the input string with accents and case removed; following that is a secondary part for the accents; following are the tertiary case differences.

Sort keys can be easily and quickly compared in binary form. Comparing only the primary parts of two strings effectively ignores differences of accent and case. By comparing only the primary and secondary parts, one can be accent sensitive but still ignore case.

So an approach suggests itself for solving the XTF's encoding and sensitivity problems: transform the input documents to UCA sort keys, and store these keys in the index. Likewise, query terms would be converted to sort keys, and these could then be looked up in the index.

This is attractive, but there are several unfortunate downsides:

- Sort keys are by definition locale-specific. The UCA supports many localized "tailorings" to support different sort orders for different locales and language groups. These tailorings produce different sort keys. For instance, in Swedish, "ä" is considered a separate letter of the alphabet, sorting after "z", while in German it's viewed as an "a" with an umlaut, sorting between "a" and "b". Choosing any one locale (including no locale) produces an index that will be a little bit incorrect for a large number of people.
- XTF would need to its own tailorings to achieve the search effects we need. For instance, in the UCA all of the following characters map to different top-level sort keys, whereas we would like them to all map to the same character: U+02BC MODIFIER LETTER APOSTROPHE, U+02C0 MODIFIER LETTER GLOTTAL STOP, U+0294 LATIN LETTER GLOTTAL STOP, and U+0027 APOSTROPHE. Luckily, adding tailorings isn't hard.
- XTF will need to apply transliteration filters in addition to sort keys, for instance in the case of Han and other non-decimal number systems, and probably mappings for Cyrillic vs. Latin.
- Sort keys are not human-readable, and this would make XTF more difficult to debug and maintain. Since sort order isn't an issue, it might be possible to map the top-level keys back to readable characters, but this is certainly not part of the UCA.

All is not lost however. The concepts and translation tables of the Unicode Collation Algorithm could very well serve as a good foundation upon which to build XTF's indexing system. One promising avenue would be to start with the UCA, add additional tailorings and transliteration, and use this for an internal index key that the user never sees. XTF would use this to get narrow down the set of terms the user might be interested in, then use locale-specific and user-configurable comparison to arrive at the exact set of terms to query.

Recommendation: Explore using the concepts and data tables of the Unicode Collation Algorithm as a starting point for XTF's internationalization features.

3. NON-LATIN SCRIPTS

This chapter will briefly cover most of the issues that will be encountered indexing and searching texts in non-Latin scripts. These include Hebrew, Arabic, Syriac, Thaana, Devanagari and other Indic scripts, Chinese, Japanese, Korean, and others.

3.1 Process Issues

The first issue to overcome is more administrative than technical: to make XTF work across a wide range of foreign language texts, we'll need to obtain sample texts, and more importantly, people who can read those texts, formulate appropriate queries, and validate XTF's responses.

Most likely this will involve collaborating with people in other countries interested in implementing XTF, or alternately, locating fluent students or testers locally.

However, the initial process of testing multi-language support in XTF can be simplified by using a simple pseudo-language based on English (such as "pig latin", or "ob"). This allows an English speaker to test the underlying functionality and verify that the system is working properly. Having native speakers becomes an issue later when verifying correct handling of stop words and word breaking.

Recommendation: Perform initial development and testing using a pseudo-language based on English.

3.2 Transliteration

Here again, the problem of transliteration rears its head. Consider the case of a student fluent in Mandarin, sitting at a public terminal in California, and hoping to access a hypothetical body of Chinese-language texts indexed with XTF. Many public terminal managers may discourage installation of any outside software. So how can this person, lacking a Chinese keyboard or proper software, effectively query the books?

One answer is transliteration: the student types the Latin letters "ching" and gets hits on the Chinese ideograph *ching*.

It's unclear how much work such a system would be, and also whether it would be of much benefit. Melvyl does some transliteration of Greek and Cyrillic (for historical reasons), and it appears that automatic transliteration has been applied to the titles of Chinese texts, though this was not done by Melvyl itself. See section 2.3 for more information on the difficulties of transliteration.

Recommendation: Evaluate available transliterators for possible inclusion into XTF's automatic indexing facilities.

3.3 Right-to-Left and Vertical Text

To XTF, the directionality of text is entirely inconsequential. Unicode defines that all text be stored in "logical order," that is, the order a person would normally read it. XTF leaves the display order entirely up to the user's web browser.

Recommendation: Nothing needed.

3.4 Interlinear Annotations

Japanese (and Chinese) books sometimes have "interlinear annotations," which essentially provide phonetic cues next to symbols which might not be familiar to the reader. These interlinear annotations (or "*ruby*" as they're called in Japanese) often convey essential

context information. This is commonly used in children's books, and also newspapers to help the reader know how to pronounce a word or phrase, or sometimes the meaning. A person may only know the kana or phonetic representation of a particular name, and if the *ruby* is there and is not indexed, then the user may not be able to find their terms.

Recommendation: XTF should index these annotations.

3.5 *Word Breaks*

In some scripts, determining the boundaries between words is not a simple as looking for whitespace. Luckily, Java includes support for determining word breaks on a locale-specific basis. The Java system will likely work for Arabic and Hebrew, but it cannot handle Thai/Lao/Khmer (which are exceedingly difficult to divide.) However, the International Components for Unicode (ICU) include a break iterator that does handle these languages.

The ICU and Java break iterators also handle the underlying punctuation cues for switching between all "simpler" languages such as Western European, Arabic, Hebrew, Ethiopic, etc.

Nevertheless, it may be necessary to implement a system of language tagging at index-time. Essentially, the index prefilter stylesheet would tag sections as to their language, and the indexer would switch word break iterators based on those tags.

On the user-interface side of things, we need a way to detect the language of the user's query. Certainly the characters they type give excellent clues, but there should also be a way for the user to override auto-detection by using a URL parameter or other mechanism. In any case, XTF would use the clues or explicit definition to select the correct word break iterator.

The ICU and the Java break iterators also have options to handle combining characters, zero-width joiners and non-joiners, and surrogate characters. Since all of these combine with previous characters to compose "text elements", the break iterators will consider them part of the previous word.

Recommendation: Use ICU's word break support, and implement language tagging if necessary. Implement automatic language detection on query strings, and a method for the user to override the automatic selection.

3.6 *Numbers and Symbols*

In addition to the European set of digits **0** through **9**, Unicode includes an additional 16 sets of decimal digits. For instance, Arabic has two sets of numerals, each of the Indic scripts has its own set, and Thai, Lao, Tibetan, Myanmar, Khmer, and Mongolian have their own digits. In each case they correspond to and are used just like the European digits. Users might want to search using European digits and expect to find numbers written many different ways.

The Han system of Chinese writing used in East Asia includes its own system of writing numbers, additive instead of the Western multiplicative system. In addition, each numeral and multiplier has alternate representations used in different situations. Tamil and Ethiopic numbers are similar. Perhaps these numbers should be translated to base 10 Latin digits, and indexed both in their original form and in base 10.

For instance, if someone is looking for the 282 codes of Hammurabi, they should be able to find 282 in Arabic (western numbers), Indic, Hebrew, as well as Babylonian cuneiform (available in Unicode 5.0).

There are a number of Unicode symbols from non-Latin scripts that users might want to search for (examples of Latin searchable symbols include **$**, **%**, and **£**.)

Recommendation: Transliterate non-European numbers to European positional digits, and index each number both ways (or at least allow searching both ways.) Add appropriate non-Latin symbols to be considered as discrete searchable "terms" in XTF.

3.7 Presentation Forms

Arabic has alternate letter forms in Unicode for use in different circumstances, such as at the start of a word, mid-word, and at the end of a word. East Asian texts have sets of full-width Roman alphabetic letters that fill an entire space normally occupied by an ideograph, as well as half-width forms. Additionally, there is a presentation set of Hangul jamo used to write Korean. In all these cases, Unicode normalization doesn't take care of mapping these presentation forms, but users will probably expect to be able to search the text by typing the root forms.

Again, Unicode "Sort Keys" provide a partial solution for this problem. See section 2.5.

Recommendation: Map the presentation forms to the root Unicode values.

3.8 Hiragana vs. Katakana

Written Japanese uses two parallel syllabaries, Hiragana and Katakana. Both can be used to write any given word, but foreign words are generally written in Katakana. There are many exceptions, and writing a native word in Katakana can be analogous to putting "quotes" around an English word, or used to convey "hipness", or may even be used to write people's names. Performing automatic transliteration for these cases would probably confuse more than aid.

Recommendation: Do not transliterate between Hiragana and Katakana. However, we may wish to transliterate Kana into Latin and enable the user to search for either the original or the transliterated versions.

4. NON-ENGLISH USER INTERFACES

Finally we turn our attention briefly to problems likely to be encountered presenting XTF user interfaces in various languages.

4.1 Locale

Java has extensive support for "locales", which are essentially sets of rules on how to break up words, collate sets of strings, etc. XTF should be modified to take advantage of Java's locale support. Perhaps it would be sufficient to extract or detect the user's locale based on

the HTTP header, but in order to support a user accessing XTF through a friend's computer, XTF should also allow a URL parameter to override this detection.

XTF supports sorting the ***crossQuery*** result set, and this sorting should observe the selected locale. For instance, in Spanish, **ch** is considered a single letter, which sorts after **c** and before **d**. For example, **charro** sorts after **como** but before **donde**. There are many other cases of locale-specific sort order (Nordic, German DIN, Turkish, Indic, Khmer, and Thai are all interesting.)

The **?** wildcard search character should also observe the current locale. Again, in English **?arro** would not match **charro**, but in Spanish it would.

Using a variant of Unicode "Sort Keys" may help; see section 2.5 above.

Recommendation: Add URL parameter to override locale, and possibly add local detection if not overridden. Propagate locale impacts to sorting and wildcard handling.

4.2 Switchable Language Result Pages

It is desirable to support user interfaces in many languages without having an entire set of crossQuery or dynaXML stylesheets for each language. It appears that it would be fairly easy to develop a set of language-independent stylesheets that access a document containing specific strings and messages based on the current locale.

The community at large can be quite valuable in this area. Mike McKenna recommends breaking the message strings used by XTF stylesheets and Java code into a single localizable file, and classifying these messages into "high", "medium", and "low" priority blocks. This allows supporters fluent in other languages to focus on the most important translation projects. In his experience, this will result in more and better translations that CDL could hope to pay for.

Also, in McKenna's experience XTF will be able to get away with a single stylesheet for virtually all Western languages, but will need separate stylesheets for each of the various East Asian locales. Even Arabic and Hebrew can use the Western stylesheet, as long as one codes the "DIR" HTML directive for right-to-left rendering as part of the loadable style attributes. In general, the biggest issues will be font size (bigger in Asia), emphasis (there are no italic ideographs), and color/branding.

Recommendation: Write language-independent stylesheets. Break out localizable messages from Java code and XSLT stylesheets into a prioritized message file, and encourage bilingual users to translate messages.

4.3 Accent Sensitivity

In many western European languages, XTF's default behavior of ignoring diacritic marks when indexing and searching suffices, and allows users who may be on an unfamiliar keyboard to still find what they're looking for.

However, these diacritics are absolutely critical to many languages, and users may want to query by specific combinations, if they know how to type them on the keyboard they have.

Thus, XTF will need to be able to accent sensitivity on or off at query time (right now it can only be switched at index time.) Unicode “Sort Keys” may help; see section 2.5 above.

Recommendation: Implement accent sensitivity switchable at runtime.

4.4 Query Parsing

XTF currently does a first-pass parse on a query as crossQuery receives it, picking out tokens and phrases. This behavior will have to be modified to be locale-specific.

In particular, many languages have their own system of quotation marks, and XTF should still pick these up as marking phrases. In addition, English users may be able to type “fancy quotes”. XTF should be able to handle all of the following:

"English" or “English”
« Français »
„Deutch“ or »Deutch«
”svenska” or »svenska»

The process of breaking the query string up into “tokens” should also take advantage of Java’s built-in locale-specific word break support. This is made fairly simple by querying the Unicode character database, which includes attributes marking quote characters.

Recommendation: Add flexible support for various styles of writing quotations.

CONCLUSION

This paper has discussed many issues and potential resolutions that will be encountered as and if CDL expands XTF to include texts written in more and more of the world’s languages. Undoubtedly there will be other problems not yet recognized, but it is hoped that all the major categories have been covered.