COSC345 Week 6 Notes

No-one should have any difficulty finding their own examples for this pair of lectures. The Swedish example might well stand; a native speaker of English might puzzle out (Hjalp/help, oss/us, ?, ?, ?, miljo/milieu) some of the words, though not enough to be useful, and I'd expect a native speaker of Arabic with no prior exposure to Swedish to be confused. Or they could try something in a familiar script but a different language, perhaps Turkish or Malay written in Arabic script. But I would expect students in Oman to already be much more sensitive to this issue than New Zealand students. At least we find the American language to be close enough to English to be intelligible, and the only major difference is the way dates are written.

Even here, it can be a problem. About 2010 my elder daughter was playing "maths games" on a web site recommended by her school, and ran into trouble with a "making change" exercise. The game presented sums of money which you were required to make up by clicking on pictures of banknotes and coins. *American* banknotes and coins. It was easy enough to figure out what a $1 note was, because it was labelled $1. (We don't have a $1 note any more, and when we did it was brown, not green.) But the coins threw her. Our coins used to go 1, 2, 5, 10, 20, 50, and now go 10, 20, 50, 100, 200. The American pattern of 1, 5, 10, 25 (and a 50 that they never seem to mention but does actually exist) was entirely novel, and the fact that the 10 cent coin is smaller than the 5 cent one caused much confusion. If the coins had been labelled with their values as well as their pictures, she might never have noticed that they were unfamiliar.

These days I really ought to mention "globalisation" (G11N) as well as internationalisation (I18N) and localisation (I10N).

Globalisation = **thinking** about the global market as part of business planning and seeking local input and taking it seriously. (Example: a company that paid a large amount to have 5 out of 30 manuals translated into Japanese, and only found out accidentally that the one that their Japanese customers really wanted wasn't one of those 5.)

Internationalisation = **developing** software so that peculiarities of the developers' culture aren't wired in; everything you might need to vary for different local markets should be replaceable. (Mac OS "resource forks", copied by Windows, and Java "resource bundles"; UNIX "message catalogues".)

Localisation = **adapting** internationalised software to a particular local market (culture, locale, *etc*).

Frankly, this set of lecture notes is mainly intended to make students aware of the problems, and is not expected to equip them to deal thoroughly with the issues. For example, I've said nothing whatever about writing documentation for translation. It turns out that writing documentation that you really expect to be translated is not the same thing as writing documentation for one local market only. Shorter simpler sentences, controlled vocabulary (say 2000 different words rather than 20,000) shared across a number of projects (for example, both Apple and Microsoft have lists of terms with standard translations), avoiding or at least glossing local allusions, all of these things can

1. reduce the cost of translation
2. improve the quality of the result
3. make the original material easier for customers

Here are some things that New Zealand students may not have needed to think about, but that people in Oman will be painfully aware of:

1. not everyone uses unaccented roman letters
2. not everyone uses American or New Zealand money (dollars and cents)
3. not everyone uses the American way of writing dates
4. not everyone uses the English way either.
5. not everyone uses the Gregorian calendar for everything
6. not everyone writes left-to-right
7. not everyone can put internal capitals in identifiers in

   thisVeyUglyStyleThatIsPopularInJavaForSomeStrangeReason

   because some scripts don't *have* capital letters

and so on. For Oman, the point of these lectures will instead be pointing out that there are ways to deal with this.

Also to point out that they are not without their problems. If you do not already have a copy of some edition of the Unicode book (on-line at http://www.unicode.org/versions/Unicode10.0.0/), get one and read through it at least once before delivering these lectures. You will find different things to complain about For example, "There are many complications in the shaping of the Arabic letter yeh. These complications have led to the encoding of several different characters". Or the fact that there are two blocks of "Arabic presentation forms" as well as the characters you are supposed to use for Arabic, so programs dealing with Arabic have to cope with multiple possible encodings of the same word. However, Oman can produce examples of that much more easily than I can.

For handouts, I used to provide copies of some pages from the Unicode book, and manual pages for the i18n/l10n stuff you find in C: date & time conversion (strftime, strptime), locales (setlocale, localeconv), converting numbers for money (strfmon), wide character reading and writing (getwc, putwc) and classification (wctype) *etc.* This stuff is all available on-line, and I've been asked not to print so much, but yes, you *should* read these manual pages. I point out that while you can get the right character for decimal point and thousands separation, you cannot use this interface (yet) to ask for real Arabic digits instead of the Western adaptation of them.

Others might prefer to give a quick tour of the international support in Java, originally from Taligent. In particular, while there's no standard support for the Islamic calendar(s) in Java, there is in the "International Components for Unicode, Java version" ICU4J, which comes from IBM. There's now an ICU web site. See http://icu-project.org/icu4j_faq.html for the Java ICU Frequently Asked Questions. In particular, I expect the question "3. Do you really support the true lunar Islamic calendar?" to be of interest to Oman.

I am slowlhy adding international calendar support to my own Smalltalk library. I've been reading a lot and thinking a lot about calendars. It's surprising what you know that isn't true. For example, I was told years ago that the Orthodox countries like Greece and Russia had finally switched over to the Gregorian calendar in the 20th century. If I've correctly understood what I've read in Dershowitz and Reingold and on the Web, it's no such thing. They've switched over to the Revised Julian Calendar, which will agree with the Gregorian calendar for several hundred years, but then they'll drift apart. (And at that, most of them still use the Julian calendar to determine the date of

Easter.) My main reference has been "Calendrical Calculations" by Dershowitz and Reingold, and if you don't have a copy, it's worth getting one for your library. There are two problems with supporting "the" Islamic calendar. The real one is observational; it doesn't depend on when the moon should be seen by someone's formula but on when it *is* seen. So it is technically impossible to get it absolutely right. There are several variants in use that can be implemented. Dershowitz and Reingold describe one based on astronomical calculations, and also two variants of comparatively simple arithmetic version. There turn out to be eight variants, not two. And of course different countries make their own choices. ICU4J follows the Saudis, but not everyone does that. How many Omani students know not to trust the "Hijri" date calculations in Microsoft Office for religious (and some civil) purposes?

There is an assumption built into the C date/time functions that the variation between locales is not a matter of which calendar but just of which names for things like months and days. Well, in trying to implement calendars like the Persian calendar in Smalltalk, I've run into the problem that French and English transliterate the Persian month names different ways (and no two English sources seem to agree either). You really need (calendar $\times$ culture $\rightarrow$ name). It's time to look outside the C and Java standards. What does the "Common Locale Data Repository" (http://cldr.unicode.org) have to say? Find out!

By the way, the autochthonous culture in New Zealand is Māori. So I wondered about implementing the Māori calendar. A study at the University of Waikato found about 45 different month$\rightarrow$name (and day-of-month$\rightarrow$name) mappings, from different tribes. So I gave up. (There is an "official" Māori calendar, which is just the Gregorian calendar with old Māori names recycled. The real Māori calendar(s) is(are) lunar, with the new year triggered by the rise of the Pleiades, so that it gets an extra month every so often to catch up.) This brings out the point that "locales" really are not in one-to-one correspondence with countries. In Turkey there are Turks and Kurds. In the land of (some of) my ancestors, there are English-speakers and Gaelic-speakers. In this country, we have English and Māori, but some "English" speakers speak British English, some New Zealand English (mostly the same but with different vowels), some Australian English, some American English (quite different accentuation, *e.g.,* someone who plays the piano is a **pi**anist here but a pi**an**nist there), some South African English, *etc.* The English you find in newspapers here has a surprisingly large number of Māori words in it. But there are regional variations in Māori as well. For example, you might find a locale `mi_NZ` for "New Zealand Māori", but North Island Māori and South Island Māori didn't even have the same repertoire of consonants. ("Otago" is a South Island name which would be "Otakou" in a northern mouth. Lake Waihola is near Dunedin, but there's no "l" in North Island Māori.)

Natural language text is a particular problem. Languages that are used in many countries are not precisely the same everywhere. I once served as a translator for an American and a Nigerian. They were both speaking English. In fact they were both speaking formal English. I could understand both of them, but they couldn't understand each other. What if they had been speaking informally? I might not have understood either of them. At another University, I once observed two Indian lecturers, both speakers of "Hindi", talking to each other in English, because that was the only way they could understand each other. Now in Arabic you have Classical Arabic, "the pure language of the

Quraish", and you have Modern Standard Arabic, and you have the vernaculars of various countries, which are even more different than the Englishes. You have to make a conscious decision, in such a case, to choose a language level (British English, say, or MSA), write to it, check repeatedly that you have written to it, and check with customers from various actual or potential markets that they understand you. This problem is particularly acute when the wide spread natural language is one like Swahili or Indonesian or English or Russian or Spanish that is a second language for many of its speakers.

Once again: it would take a whole paper to inculcate the beginnings of skill in developing internationalised software. These two lectures are only supposed to make students aware that the problem exists and that something can be done about it. But a maintenance project that involved internationalising some existing program (even something like the Portable C Compiler, which has recently been revised for BSD) and then arabising it would be a very interesting thing to try.

By the way, I found Australian lawyers using the minus/hyphen as their decimal point. Presented with a bill for "58 – 99" I couldn't make them understand why that meant they owed *me* money. This counts as another "locale", but there is no official locale for it. This raises another issue, which is that it may be necessary to "localise" software *below* the level to which the operating system is normally willing to go. Locale names typically mention country, language, and character set. But Australian states have different public holidays. And different professions may use different ways of indicating negative numbers. Within my own culture, a debt of 100 dollars has been variously notated as -100, (100), and 100 in red ink, which is why you used to be able to get typewriter ribbons that were half black and half red. We still speak of a person or business being "in the red".

Unicode keeps on growing. Leaving aside surrogate codes, private use areas, and code points classified as "noncharacter", here are the *increases* for each version. These figures are derived from DerivedAge.txt in the Unicode data base, which does not provide information about Unicode 1.0.

1.1	27,577	characters
2.0	11,373	more
2.1	2	more
3.0	10,307	more
3.1	44,946	more
3.2	1,016	more
4.0	1,226	more
4.1	1,273	more
5.0	1,369	more
5.1	1,624	more
5.2	6,648	more
6.0	2,088	more
6.1	732	more
6.2	1	more
6.3	5	bidorectional controls
7.9	2,834	more
8.0	7,716	more
9.0	7,500	more
10.0	8,518	more

Unicode 5.2 had 50 versions of zero. TAG DIGIT ZERO doesn't really count as that's for meta-data, not displayable digits. Only the characters labelled "Nd" are ones you might want to use in reporting numbers normally. That still leaves 41 different zeros, and no way in C to tell printf() which one to use. It does seem that the %O modifier in strftime() might select the locale's digits, though.

```
0030;DIGIT ZERO;Nd
0660;ARABIC-INDIC DIGIT ZERO;Nd
06F0;EXTENDED ARABIC-INDIC DIGIT ZERO;Nd
07C0;NKO DIGIT ZERO;Nd
0966;DEVANAGARI DIGIT ZERO;Nd
09E6;BENGALI DIGIT ZERO;Nd
0A66;GURMUKHI DIGIT ZERO;Nd
0AE6;GUJARATI DIGIT ZERO;Nd
0B66;ORIYA DIGIT ZERO;Nd
0BE6;TAMIL DIGIT ZERO;Nd
0C66;TELUGU DIGIT ZERO;Nd
0C78;TELUGU FRACTION DIGIT ZERO FOR ODD POWERS OF FOUR;No
0CE6;KANNADA DIGIT ZERO;Nd
0D66;MALAYALAM DIGIT ZERO;Nd
0E50;THAI DIGIT ZERO;Nd
0ED0;LAO DIGIT ZERO;Nd
0F20;TIBETAN DIGIT ZERO;Nd
1040;MYANMAR DIGIT ZERO;Nd
1090;MYANMAR SHAN DIGIT ZERO;Nd
17E0;KHMER DIGIT ZERO;Nd
1810;MONGOLIAN DIGIT ZERO;Nd
1946;LIMBU DIGIT ZERO;Nd
19D0;NEW TAI LUE DIGIT ZERO;Nd
1A80;TAI THAM HORA DIGIT ZERO;Nd
1A90;TAI THAM THAM DIGIT ZERO;Nd
1B50;BALINESE DIGIT ZERO;Nd
1BB0;SUNDANESE DIGIT ZERO;Nd
1C40;LEPCHA DIGIT ZERO;Nd
1C50;OL CHIKI DIGIT ZERO;Nd
2070;SUPERSCRIPT ZERO;No
2080;SUBSCRIPT ZERO;No
24EA;CIRCLED DIGIT ZERO;No
24FF;NEGATIVE CIRCLED DIGIT ZERO;No
A620;VAI DIGIT ZERO;Nd;0;L;;0;0;0;N;;;;;
A8D0;SAURASHTRA DIGIT ZERO;Nd
A8E0;COMBINING DEVANAGARI DIGIT ZERO;Mn
A900;KAYAH LI DIGIT ZERO;Nd
A9D0;JAVANESE DIGIT ZERO;Nd
AA50;CHAM DIGIT ZERO;Nd;0
ABF0;MEETEI MAYEK DIGIT ZERO;Nd
FF10;FULLWIDTH DIGIT ZERO;Nd
104A0;OSMANYA DIGIT ZERO;Nd
1D7CE;MATHEMATICAL BOLD DIGIT ZERO;Nd
```

```
1D7D8;MATHEMATICAL DOUBLE-STRUCK DIGIT ZERO;Nd
1D7E2;MATHEMATICAL SANS-SERIF DIGIT ZERO;Nd
1D7EC;MATHEMATICAL SANS-SERIF BOLD DIGIT ZERO;Nd
1D7F6;MATHEMATICAL MONOSPACE DIGIT ZERO;Nd
1F100;DIGIT ZERO FULL STOP;No
1F101;DIGIT ZERO COMMA;No
E0030;TAG DIGIT ZERO;Cf
```

The lost characters in English are ash (æ Æ), eth (ð Ð), thorn (þ Þ), yogh, and wynn. Modern English has a pressing need for esh (∫ Σ), eng, and either eth or thorn. ("Þe cat sat on þe mat" — thorn. "Ðe cat sat on ðe mat" — eth. "Σe sells sea-∫ells" — esh.)