Bridging the Digital Divide, the Future of Localisation

Patrick A.V. Hall,
The Open University, UK
p.a.v.hall@open.ac.uk

ABSTRACT
Software localisation is reviewed within its economic context, where making computers work in many languages is not economically worthwhile. This is challenged by looking for different approaches to current practice, seeking to exploit recent developments in both software technology and language engineering. Localisation is seen as just another form of customisation within a software product line, and the translation of the text and help messages is seen as an application of natural language generation where abstract knowledge models are built into the software and switching languages means switching generators. To serve the needs of illiterate peoples the use of speech is considered, not as a front end to writing, but as a medium that replaces writing and avoids many of the problems of localisation.

BACKGROUND
It is some 25 to 30 years since localisation issues surfaced in software, when bespoke software began to be developed by enterprises in industrialised countries for clients in other countries, and when software products began to be widely sold into markets other than that of the originator. Initially these systems were shipped in the language of their originators, typically English[^1], or a very badly crafted version of some local language and its writing system. There were no standards initially for the encoding of non-Roman writing systems, and localisation was very ad hoc. But things have changed in the intervening years. There has been the really significant development of Unicode, so that we can now assume that all major writing systems are handled adequately, and that Unicode has been made available on all major computing platforms. Unicode arose out of developments in Xerox during the 1970s and 1980s, with the first Unicode standard published in 1990. All platforms also now offer the more-or-less standard set of localisation facilities established during the 1980s. These are packaged together in an Application Programming Interface (API) embracing locales (identifiers indicating country, language and other factors that differentiate the location of use) and their management, together with various routines for handling the writing system, dates, currency, numbers etc that belong to this locale. Platforms also have low level facilities for segmenting software so that those parts of the software that change with localisation can readily be replaced during the process of localisation – these local dependent parts are placed in resource files. Books from platform suppliers about localisation only began appearing in the early 1990s with the first general book in this area by Dave Taylor appearing in 1992. The uniformity of facilities across platforms and programming languages is really quite remarkable, since this is not regulated by international standards and indeed when a proposal was brought forward in the mid-1990s it did not get support[^2].

[^1]: It is now recognised that shipping software in an international language like English is not good enough, even though English is in such widespread use. LISA suggests that as much as 25% of the world's population are competent in English, but clearly this is an over estimate – David Crystal writing in 1997 estimated that around 5% of the world's population used a variant of English as a mother tongue, and a further 3% had learnt it as a second language.

[^2]: There was an attempt around 1995 to formulate an ISO standard 15435 for an internationalisation API – the draft relied heavily upon the facilities available in Posix, and did not progress through lack of support from the wider programming languages community. This is regrettable since an abstract interface could have been formulated with bindings to particular programming languages and platforms. This means that simple plug compatibility across platforms and programming languages is not guaranteed.
Localisation of software also emerged as a distinct industrial practice during the 1980s. Localisation began to be outsourced, and pockets of expertise, like that in the Dublin area, emerged. The Localisation Industries Standards Association (LISA) was founded in Switzerland in 1990 by Mike Anobile, and has grown every year since. Today LISA sees the objectives of localisation as embracing not just linguistic issues, but also content and cultural issues and the underpinning technologies – “modifying products or services to account for differences in distinct markets” (LISA p11).

**LOCALISATION ECONOMICS**

Localisation (also referred to as L10N by technical specialists) is seen by LISA as “not a trivial task”, but what localisation costs as a proportion of the original development cost is not clear. It is common practice in the software industry to relate additional costs, like post delivery maintenance and re-engineering to improve maintainability, to the original development cost. So for example the planning norms at a large installation I worked at recommended resourcing the first year of maintenance at 20% of development costs, and successive years at 10%. Harry Sneed, an authority on re-engineering, has reported how following a scheduling busting development which left the software working but totally unstructured and undocumented, he won a contract for 20% of the original development costs to re-engineer the software and make it maintainable. Just what proportion of original development cost is required for localisation? I would guess of the order of 10%, but have no good basis for that guess.

It is generally agreed that software should be designed so that subsequent localisation is relatively cheap. This design-for-localisation is called internationalisation (also referred to as I18N), and may be done during original development, or as a stage following development. Sometimes internationalisation is also known as globalisation, though globalisation is also used to refer to the round trip of internationalisation followed by localisation. “A good rule of thumb to follow is that it takes twice as long and costs twice as much to localize a product after the event” (LISA p12). There clearly are very good economic reasons for internationalising the software.

So what does globalisation cost? Internationalisation seems to halve each subsequent localisation step, but how many localisation targets do you need to make an internationalisation reengineering stage cost effective? We don’t seem to know. But what we do know is that the cost of localisation, even following internationalisation, can be significant, so significant that localisation for many markets may just not be worth while, or only warrant the most rudimentary localisation.

During localisation the bulk of costs - around 50% - go in translation of the various text messages, menus, help, documentation etc, though clearly the exact balance depends upon the extent of localisation involved (LISA 1999). It is the objective of this paper to show how by adopting suitable technical strategies, the marginal cost of localisation can be reduced very significantly, making localisation to even relatively minor languages and cultures viable.

**DEVELOPING COUNTRIES AND COMMUNITIES**

While commercial parties understandably are driven by profit, or at the very least by the need to cover costs, there are other important considerations to bear in mind. If we are to help countries develop, could Information Technology help? If it could, should we actively facilitate the uptake, and not leave it to commercial development and the profit motive? This has been the subject of much debate, people cannot eat computers, and yet could computers...
help? Could the vast information resources available on the Internet be useful to economically depressed communities? Could the Internet help people share development information? The barriers to this are twofold; economic, and the lack of localisation.

At the economic level computers and internet connections cost one or two orders of magnitude more relative to people’s incomes than they do in the west. In the west we earn enough in a few weeks to buy a computer, in developing countries a year’s earnings may not be enough. But unlike in the west, in developing countries people are happy to share resources. Telecentres of various kinds are being installed all over the developing world, with their success and failures regularly being reported in Internet discussion groups like GKD run by the Global Knowledge Partnership.

Much less well considered have been the barriers to use created by lack of localisation. If localisation has been considered at all, it seems to have been viewed as trivial, but this clearly is not the case. People in developing communities may not be literate, and if they are literate they may only be literate in some local national language. The facilities of computers, like browsers, as well as digital content, need to be available in the person’s own language, in writing if the language is written, but also in speech. To illustrate, in Nepal the official language is Nepali, written with a variant of the Devanagari writing system used for Hindi. Education over most of the 50 years of universal education has been in Nepali, so that today nearly everybody speaks Nepali, though only some 30% are literate in Nepali. About half the population would claim that Nepali was their first language, the other half speaking one of the other 70 odd languages of Nepal, many of them without written forms.

To indicate that there is a need, let us consider just two projects, Kothmale and HoneyBee. Kothmale represents an intermediate technical solution, and is built around the Kothmale radio service in Sri Lanka. At Kothmale listeners are encouraged to telephone in questions in the own language – these questions are then answered using the Internet, with the answer broadcast via the radio station. This UNESCO funded project has become an example for many other initiatives, with cheap access to the web using speech, at the cost of a telephone call and a radio, albeit mediated by humans – though it is not clear just how many initiatives have actually been taken through into operation.

The HoneyBee network (see Gupta et al, 2000) was created to share indigenous knowledge among rural communities, with an interest in patenting inventions and enabling the peasant inventors to obtain income from their invention. Originally information was disseminated in a newsletter, but this has now been replaced by a website at http://csf.colorado.edu/sristi.

NEW TECHNICAL APPROACHES
The current technical approaches to localisation, described above in the Background section, were invented nearly 30 years ago. Since then technology has advanced significantly, but localisation has not kept pace with this advance. Object oriented approaches, and in particular Java, have become widespread in use. The established methods from 30 years ago, of a localisation API using locales and Unicode and various routines, have been implemented in the latest languages, like Java. But text books (e.g. Winder and Roberts 2000) do not cover localisation – for that you must go to specialist books like Dietsch and Czarnecki (2000). By contrast books on XML (e.g. Harold 2001) typically do cover localisation in some measure, though even here specialist books are appearing (e.g. Savourel 2001). More general books on localisation are also still being published, but typically here they still refer back to earlier languages like C and C++ since the bulk of current software was developed in these (e.g. Schmitt 2000). But all this involves a technical approach that is essentially 30 years old.

Software development is now becoming component-based. Components now enable us to package together inter-related facilities, and this is the way that the localisation API’s
should be viewed. Localisation facilities should be available as components which can be replaced by some simple technical process to change locale. If appropriate this switch (relinking) could be achieved dynamically and at run time in multi-lingual working. Experience of handling internationalisation and localisation should be captured as analysis and design patterns, and be made more widely available. This has yet to be done. And all of this should be pulled together within a coherent whole using an appropriate product line approach – product lines are a series of closely related software products serving a very closely related set of customers or markets, see for example Jan Bosch (2000). We normally think of product lines being closely related applications, like software to control motor vehicle engines, where the functions remain essentially the same and the software varies as a result of the particular engines it must control. Yet this concept works equally well for localisation, but do we think of it in this way?

Applications need to move beyond resource files as offered on platforms for application segmentation and the incorporation of product variants. It may simply be a matter of re-expressing and re-packaging existing technologies (so for example resource files are viewed as classes in Java), but it may require more radical advances to software platforms to enable this. Such advances would be similar to that taken to move platforms from simple single-byte views of character codes to embrace Unicode.

Usability engineering has taken a much more prominent position in software development, driven by many failures of software in use (see for example Landauer 1995). Usually the remedy is to involve potential users more intimately in the software development process, but a deeper analysis of user needs is also required. Localisation is but one further extension of this idea of enhancing usability to increase system acceptance. We will look at four aspects of this, language, literacy, culture, and business.

**LANGUAGE**

We saw above that translation costs account for around half of localisation costs. If we are looking to make our software systems accessible to many more linguistic groups, this translation cost is going to dominate. Can anything be done about this?

There are vastly many more languages in the world than are acknowledged in Unicode. Exactly how many is a complex issue, as one separates dialects from languages, and the various names for languages from the languages themselves. Nettle and Romaine (2000) judge that there are between 5,000 and 6,700 languages world-wide, most of which are not written, nor even described in academic literature. However most societies have dominant national ‘official’ languages that are written and are the basis for national life and business – there are only a couple of hundred of these. For example India with over one billion people has 17 official languages but recognises around 380 languages in current use. By contrast, the United Nations recognises 185 nation states, but only has 6 official languages! In thinking about localising software and digital content we must not be seduced by a small set of official languages, and instead must enable ourselves to serve as many of those 6,000 or so languages as possible. A software product with a few tens of languages supported has only scraped at the surface of global outreach.

The way to handle this very large range of languages is to extend the current practice of message composition by recognising that this is a limited form of what computational linguists call Natural Language Generation (NLG). The idea of language generation is to represent the area for which messages need to be generated in some language neutral knowledge model, and then to generate sentences and longer passages of text using this model. The generator must have a suitable lexicon and an appropriate syntax for the language concerned and the domain covered by the knowledge model. See for example the book by Reiter and Dale (2000).
Changing language means switching generator. This was demonstrated in the 1990s on the EU funded Glossasoft project (see Hall and Hudson 1997). A model of the software was built and then as the user took actions that required an informative message from the system, this was generated from the model and the contingency that triggered the message generation. Messages could be created in different styles, depending upon the preferences and level of expertise of the user.

NLG has also been used for digital content, in a series of very forward looking projects at Brighton University in the UK (e.g. Power, Scott and Evans 1998). Instead of representing digital content as a body of text, it is represented as a language-neutral knowledge model. Tool support enables a user to develop the required knowledge model without being a knowledge engineering expert. Using meta-knowledge the tool guides the user in making choices, which are then presented to the user in natural language using natural language generation. This can be made multilingual by incorporating other generators, with the potential for multiple authors creating digital content together using different languages. The HoneyBee Network referred to above, for example, could benefit enormously from this technology.

The potential here is that the same generator should be usable in many different systems, thus spreading the cost. However I emphasise the word “potential” – over the past ten years or more there has been the systematic sharing of linguistic resources within Europe, mediated by ELRA, the European Language Resources Association. This is operated commercially to industry developing multi-lingual products, but has also allowed the free exchange of resources within the language engineering research community. While these resources do aim to conform to standards developed within Europe, there have been some difficulties in picking up and reusing the resources such that some researchers have just developed their own resources. So far ELRA has not aimed at supporting the localisation industry, but there is significant commercial potential here.

LITERACY AND SPEECH
In many societies literacy is relatively low, often less that 50%. Using speech to access computing facilities and content is highly desirable – we touched on this earlier in referring to the Kothmale project.

Systems exploiting speech processing are becoming more and more common. Dictation software is now really very good, and with only a small level of training before people can dictate documents with a high level of accuracy. Speech dialogue systems are also operational in a number of telephone-based enquiry systems – see for example Bernsen et al (1998). It is clear that we can adapt our software systems to work in speech using well established technologies, with one significant proviso – current speech systems are mediated by the written form of the language in subtle ways, and we must move beyond this.

This dependency on written language is most noticeable with dictation systems. In principle you could view these as enabling you to compose speech documents, but in order to navigate around the document, and to move material, you need to interact with the written form of the document using the normal text editing functions of a word processor. We need some way of interacting with the document that does not require the ability to read. Roger Tucker at HP Labs in the UK developed a prototype for a pure speech personal organiser which includes some reasonable capability for visualising and editing speech documents, though a richer set of facilities is required – Alan Blackwell has termed this facility “Speech Typography”. We must move away from the written form of the language and work solely with the spoken form and its encodings in the computer. This is very challenging: in effect we need to recreate for speech what has taken 3000 years to develop for writing.
Speech technology to date has, naturally, focussed on the languages of major industrialised nations, and does need to broaden its outreach to cover the very wide range of spoken languages of the world. The attractive thing about pure speech systems is, however, that they are in general language neutral.

**Cultural Abstractions**

As LISA has emphasised, language is only a part of the problem, albeit a large part given the translation load it generates. We already do a lot about cultural conventions during localisation, in handling number formats, sort orders, formats for time and dates, addresses, and similar. These are now embedded in practice through the APIs that are used. But we need to do much more.

A range of other cultural conventions need to respected. Calendar systems other than Gregorian are not well handled. The way people are named varies, not just in order of presentation as in the difference between East Asian names and European names, but also in what constitutes a name and the circumstances under which it is used. Colours have different significances depending upon culture, so for example red may mean danger in Europe and marriage and happiness in China. Mourning is denoted by black in Europe, but white in South Asia. Icons are cultural specific, yet the meaning they are intended to convey is determined by the application. Some cultures like cluttered and busy screens, others like them sparse and minimalist. Members of some cultures like the computer to instruct them what to do, members of other cultures want to be in control of the computer.

Can we make some abstraction of these which enables us to switch cultures as easily as we can switch languages? We could easily imagine a set of standard meanings where icons are typically used, with the actual icons changing as we switch locales. Similarly we might wish to colour some message or its enclosing box with a colour that signalled danger, and have this change as we changed locale. At the moment we cannot even make these simple switches, let alone use an array of emotive colours or shapes that vary with locale. Of course choice of colour and shape are just simple aspects of screen design, and while design is determined by the encompassing cultures and its conventions and aesthetics, maybe we do have to accept that where design is important each new locale will justify a new design.

It is tempting to characterise cultures by some simple set of parameters, and use these parameters to drive interaction choices as locales are switched. Geert Hofstede (1991) reported a very large multinational study which arrived at just four dimensions of significant difference between cultures: individualism versus collectivism, autocratic versus democratic organisation (power distance), assertiveness versus modesty, uncertainty acceptance versus avoidance. Marcus and Gould (2000) have analysed websites from this perspective to give an account of the differences observed between web-sites in different cultures. However others (El-Shinnawy and Vinze 1997) have shown that the use of simple cultural parameters cannot be used to predict user behaviour. Simple cultural parameters cannot be used as a basis for the cultural abstractions in software that could be switched as locale is changed. It is clear that obtaining simple cultural abstractions are possible and should be taken, but that any comprehensive characterisation of cultures may never be possible.4

**Business and Organisation**

Businesses and organisations in the same general area of activity, like hospitals or insurance companies, do very similar things, and need very similar software systems to support them. We can abstract the common ground, and produce generic systems which can be specialised

---

4 ISO is in the process of adopting a standard, ISO/IEC 15897, for registering cultural profiles where most aspects of the culture are described in text, though set out beneath standard headings.
for particular customers. This is the approach of product lines and product families discussed above. It is also the basis for the success of ERP systems, although the degree of abstraction and genericity in these can be very limited, such that they are not truly product line approaches. Other attempts to produce an industry wide generic capability, like IBM’s SanFrancisco project, are rumoured to have failed.

How generic can we be? We know that we can isolate particular aspects of law, like taxation, and thus make financial systems transportable across markets. But could we abstract more general legal principles and build software around that – for example, could we build an abstract model of European employment law, and its embodiment in various national legal systems, and then use that to parameterise Human Resource Management Systems?

CONCLUSIONS
We have seen that we can address smaller linguistic and cultural markets for software products, and significantly increase access to information technology. This can be achieved by reducing the marginal cost of localising software and content to a new language and culture. This must be paid for by developing reusable resources and obtaining agreements so that the costs of developing these resources can be spread over many uses.

For languages this means moving to embedding the meaning of messages and interactions within the software, using natural language generation technologies to create messages that output this meaning to the human user. Speech input and output will be important for people with low literacy levels, and methods of handling spoken language free from written forms need to be developed. For more general cultural and business features, this means seeking general abstractions of these features that are as widely applicable as possible, though we cannot expect universal models of culture. We may well need a number of distinct abstractions and conventions representing different groups of languages and cultures.

Replacement of one language and culture by another means substituting one software component by another. Overall coherence is assured by taking a product line approach to software development. To make this possible we will need well defined interfaces which are commonly agreed to. Regulation of these interfaces through international standards organisations would be appropriate.

All this needs further research and development, focusing on the key areas outlined above. This range of research and development problems are being further explored within the EU-funded SCiLaHLT5 project, with particular problems being addressed in other projects. There is a need for much further work to move this vision into reality.

ACKNOWLEDGEMENT
I would like to acknowledge support over many years from the UK EPSRC and the European Union in carrying out the studies that underpin this paper. In particular I was supported by the EU Asia IT&C project ASI/B7-301/97/0126-05 ‘SCiLaHLT’ to present this paper at the ITCD conference in Kathmandu in 2001.

REFERENCES

5 The Sharing Capability in Localisation and Human Language Technologies (SCiLaHLT) project is funded by the EU under its Asia IT&C programme. It focuses on South Asia, and aims to help aid projects use localised IT&C systems to disseminate development knowledge.

Bosch, Jan (2000) *Design and Use of Software Architectures*. Addison Wesley.


Nettle and Romaine (2000) *Vanishing Voices, the extinction of the world’s languages*. Oxford University Press.


