

Why Human Language Technologies are critically important for Bridging the Digital Divide

Briefing notes for SCALLA 2001¹, Bangalore, 21st to 23rd November 2001. Patrick A.V. Hall, The Open University, UK

Background

Let us start by claiming that information technology, and in particular the Internet, should be available to all peoples in their own mother tongue, whatever that might be, however small the population of mother tongue speakers might be. Information technology must move to meet the user community and provide all its services fully localised, working in the local languages and according to local cultural conventions. That this is desirable is not universally shared.

Is the localisation of IT to all languages and cultures, however small, a reasonable ideal objective?

Communities using localised information technology will be enabled by information technology to develop their communities in whatever way they see appropriate. But today we live in a globalising world, with the expectation that if peoples are to prosper they must participate in the global economy. Information technology is seen as a critical ingredient of this new global economy (eg Cassels 1996). The global and the local could be in conflict. James Wolfensohn, the President of the World Bank, in his foreword to the 1999-2000 World Development Report "Entering the 21st Century" writes:

"Globalisation is praised for bringing new opportunities for expanded markets and the spread of technology and management expertise, which in turn hold out the promise of greater productivity and a higher standard of living. ... Localisation is praised for raising levels of participation and involvement, and providing people with a greater ability to shape the context of their own lives. ... Rather it recognises them as forces that bring new opportunities but also raise new or greater challenges in terms of economic and political stability."

I would argue that technology can help to bridge the gap between local communities and the global economy. In this working conference we will explore what technology must be put in place in order for communities to prosper in diversity.

Until around 25 to 30 years software was produced for only the language of its originators, typically then English though not exclusively so. Then progressively 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 originators², 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 more-or-less adequately, and that Unicode has been made available on all major

¹ This conference is partly funded by the European Union in the Asia IT&C programme, as part of the SCiLaHLT (Sharing Capability in Localisation and Human Language Technologies) project under contract ASI/B7-301/97/0126-05

² 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.

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 facilities established during the 1980s, with an API embracing locales³ and their management, and various routines for handling dates, currency, numbers etc. 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. Books from platform suppliers about localisation only began appearing in the early 1990s and the first general book in this area by Dave Taylor appeared 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 support⁴.

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).

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? This has been the subject of much debate⁵, people cannot eat computers, and yet could they 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 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 (see for example UNESCO 2001).

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 persons 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 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.

³ A *locale* can denote any grouping of users for whom a distinct set of linguistic and cultural conventions exist. In the computer this enables us to change these conventions by the simple expediency of switching locales.

⁴ 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.

⁵ For example, the G8 raised the DOT force study, which included an internet based consultation called DIGOPP during the first half of 2001. The UK Department for International Development consulted widely for a white paper on Globalisation and Development, which included much consideration of the role of ICTs in development.

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.

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>.

Localisation economics.

Localisation 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 globalisation or internationalisation, and may be done during original development, or as a stage following development. “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 globalising the software.

So what does globalisation cost? It seems to halve each subsequent localisation step, but how many localisation targets do you need to make a globalisation reengineering stage cost effective? We don't seem to know. But what we do know is that the cost of localisation, even following globalisation, 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 thesis here that by adopting suitable technical strategies, the marginal cost of localisation can be reduced very significantly, making viable the localisation to even relatively minor languages and cultures.

New technical approaches show us the way

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.

Object oriented approaches, and in particular Java, have become widespread in use. The established methods from 30 years ago, of locales and Unicode and interaction APIs, have been implemented in the latest languages, like Java. But text books (eg Winder and Roberts 2000) do not cover localisation – for that you must go to specialist books like Deitsch and Czarnecki (2000). By contrast books on XML (eg Harold 2001) typically do cover localisation in some measure, though even here specialist books are appearing (eg Savourel 2001). Specialist 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 (eg Schmitt 2000).

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's book (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, 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.

Could a set of standard interfaces be defined for all localisation elements needed? This can only be answered when all the questions below have been resolved.

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.

Writing systems and what may still be needed.

Normally we think of a language requiring a writing system before material in the language can be usefully stored and manipulated in the computer. Communication with the human user then requires special input devices like keyboards and tablets and OCR, and output devices like printers and monitors.

Fundamental to this is the computer encoding of the writing systems or scripts of the languages. For a long time these were completely centred on the Roman alphabet and American English through the 7-bit ASCII, with some allowance for European languages in ISO 646. Then in the 1980s the multi-byte Unicode (now ISO 10646) emerged to become the encoding systems for writing systems across the world. In principle Unicode can encode all the scripts of the world though there are many shortcomings and omissions. For example, Unicode represents Indic writing systems based on an interim 1988 version of the Indian national ISCII standard (Bureau of Indian Standards 1991), and Unicode needs updating, though ISCII's view of Indic writing has itself been questioned. Unicode has no representation of the Marathi or Nepali writing systems claiming that these languages should use Devanagari which is very close. Nevertheless we can view the encoding of writing systems as largely completed, though linguists do need to monitor what the Unicode standards organisation does lest they misunderstand their writing system, and do need to make proposals to them when appropriate.

Are the writing systems of South Asia adequately represented in Unicode? How can we systematically create new codes for writing systems and languages currently not included?

With the encoding comes means of rendering the writing either on paper or on the screen. There is never a direct correspondence between computer internal codes and characters in print or display, but in some systems, like those of South Asia, the correspondence is much more complex than in the Roman writing systems. Historically these problems have been addressed in typography in the print industry (eg Ross 1999). The current development of Open Type rendering promises very high quality results. Fonts are being developed for many of the South Asian scripts, but we must anticipate much scope for further development of

fonts within the communities using the fonts (see Mudur et al 1999). A high quality font takes a person year or more to develop, even with good tools.

Are South Asian writings systems being adequately rendered, even in the latest Unicode linked systems?

Independently of the output rendering and internal coding are processes for writing into the computer. The most widespread method uses keyboards, and here immediately much confusion can arise. There is no need for the keys available to correspond to characters of the writing system represented in the code, nor is there any need for the sequence of typing (or protocol) to correspond to sequence in storage demanded by the encoding. Typically there will be keyboard layouts and methods of typing that have become established from the age of the typewriter, and these will be adopted for the computer. Nevertheless there can be persuasive reasons for adopting new layouts and protocols to facilitate typing – for example the INSCRIPT keyboard of the ISCII standard, or similar proposals for Arabic dating back to the 1970s or earlier. As an alternative to keyboards is tablet input that analyses the strokes of writing, increasingly important in hand held devices. Also of great importance is OCR input to enable the bulk capture of much existing data only currently available on paper. Both of these last two input methods may require disambiguation using knowledge of likely combinations, both lexical and linguistic.

What is currently available for South Asian languages? Are there any issues here grounded in a deep understanding of the writing system or in language?

Many languages are unwritten, but are in the process of having writing systems developed for them. For example India is reported to have in excess of 350 distinct languages, of which less than 50 have established writing systems. The new writing systems will be based on some existing system, very likely an alphabetic system. For Asia the system will probably be based on a local system which in turn will most likely have been derived from the Brahmi systems that arose in India around 2500 years ago. In the former USSR the writing systems will have been Russian, imposed from Moscow in the mid 20th century though many countries are now reverting to earlier writing systems, or even adopting a Roman system. Elsewhere the writing system adopted (or imposed) has been Roman-based as a result of earlier European colonisation. I have heard of a case in the Terai along the border region of northern India and Nepal where the people insisted on a Roman based writing system because of the economic dominance of English.

The new writing system is constructed typically on phonetic similarity using diacritics to vary sounds where the base alphabet cannot make the distinction. This happens either for administrative reasons as in much of colonial history, or because scholars (like the Himalayan Language Foundation based in Belgium) are interested to record the language before the language ceases to be used, or to enable religious or political proselytisation (for example the US based Christian group, the Summer Institute of Linguistics, SIL). While the motives for creating writing systems must often be questioned, the net effect of enabling the writing of the language may be beneficial. The creation of a writing system will usually be accompanied by an encoding of it for the computer, since further work on the language will be computer based.

How can the construction of writing systems for previously unwritten languages be constructed systematically so that communities can be assured of a high-quality and appropriate result?

Language opportunities and challenges

Much of software localisation, and indeed content localisation, involves an activity very close to translation. 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 (the quip is that a language is a dialect with an army and a navy) 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 and to reduce the cost of adding a new language is to exploit natural language technology.

An immediate candidate seems to be machine translation, though to date for syntactic machine translation to be successful the domain must be narrow and the languages must not be too dissimilar. Even then some pre and post-editing is essential for quality translations. The alternative approach is translation memory with some generic capability. The cost of human translation is reckoned to be about halved by the use of a translation memory system.

How successful have translation systems between South Asian languages been? And what about between South Asian languages and other languages, like English?

Current practice is to store messages outside the software in resource files, and put the number of the required message in the software. In some cases we must compose messages out of parts, filling in values in slots in the stored message. We can recognise that this is a limited form of what computational linguists call Natural Language Generation (NLG). The message number embodies the intended meaning, with the indexing into the resource file with simple slots and fillers is the language generation. For a different language you just switch files. The idea in full language generation is to represent the area for which messages need to be generated in some language neutral knowledge model, and then to create sentences and longer passages of text from this model in response to some triggering event. 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 (eg. 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.

How much work on language generation is being undertaken in South Asia? Are there any aspects of the languages which make it particularly difficult or particularly easy?

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 as a free exchange of resources within the language engineering research community, and while these resources do aim to conform to standards developed within Europe, there have been some severe difficulties in picking up and reusing the resources such that some researchers have just developed their own resources. But there are also examples of very successful reuse,

such as the British National Corpus. There is here the basis for some significant commercial enterprise supporting the software localisation industries.

All this begs a number of questions about linguistics and language processing. Just how language independent can a knowledge model be? Could it at least be adequate across large classes of language, and if so do these classes relate to the usual way linguists classify languages? How domain independent can a linguistic resource like a generator be? Could we at least get transportability across some suitable range of domains?

Are there particularly kinds of model of language, like statistical models, that make these processes easier or more difficult?

Literacy and the challenges it raises for Speech

In many societies literacy is relatively low, often less than 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 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.

What speech recognition and speech generation capability has already been developed for South Asian languages? It is claimed that the writing is very phonetic, but is this really the case such that it helps in recognition and generation?

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 – Alan Blackwell has termed this facility Speech Typography, and this will be the subject of an upcoming research project. 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.

Has any work been done on pure-speech systems in South Asia?

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 be 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 cultural parameters 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.⁶

What are the key aspects of culture that should be thought about and changed during localisation? Could these key aspects be formulated as a model of culture that could be used to facilitate the switching between cultures?

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 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, though 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 SanFranciso 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 and within digital content, 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

⁶ 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.

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 SCiLaHLT project, with particular problems being addressed in other projects. There is a need for much further work to move this vision into reality.

References

- Bernsen, N.O., Dybkjaer H. and Dybkjaer L (1998) *Designing Interactive Speech Systems. From first ideas to User Testing.* Springer Verlag.
- Bhatnagar and Schware (Editors) (2000). *Information and communication technology in development. Cases from India.* Sage.
- Bosch, Jan (2000) *Design and Use of Software Architectures.* Addison Wesley.
- Castells, Manuel (1996) *The Rise of the Network Society. Volume 1 of The Information Age: Economy, Society and Culture* Blackwells
- Crystal, David (1997) *English as a Global Language.* Cambridge University Press.
- Deitsch A. and Czarneci D.A. (2000) *Java Internationalization,* O'Reilly, UK
- EI-Shinnawy M. and Gould A.S. (1997) Technology, culture, and persuasiveness: a study of choice-shifts in group settings. *International Journal of Human-Computer Studies*, 47, 473-496
- Gupta A.K., Kothari B. and Patel K (2000) Knowledge Network for Recognizing, Respecting and Rewarding Grassroots Innovation. Chapter 8 in Bhatnagar and Schware (2000).
- Hall P.A.V. and Hudson R. (1997) *Software without Frontiers.* John Wiley & Sons.
- Harold E.R. (2001) *XML Bible,* Hungry Minds.
- Hofstede, G. (1991) *Cultures and Organisations. Software of the Mind. Intercultural Cooperation and its Importance for Survival.* . Harper Collins.
- Landauer Thomas K. (1995) *The Trouble with Computers. Usefulness, Usability and Productivity.* MIT Press.
- LISA (1999) *The Localisation Industry Primer.* Localisation Industry Standards Association, Geneva.
- Marcus A. and Gould E.W. (2000) Crosscurrents: Cultural Dimensions and Global-WebUser-Interface Design. *ACM Interactions*, |VII (4) pp 32-46.
- Mudur, S.P., Niranjan Nayak, Shrinath Shanbhag, R.K. Joshi (1999) An architecture for the shaping of Indic texts. *Computers & Graphics* 23. pp 7 – 24
- Nettle and Romaine (2000) *Vanishing Voices, the extinction of the world's languages.* Oxford University Press.
- Power R., Scott D. and Evans R. (1998) What You See Is What You Meant: direct knowledge editing with natural language feedback. In Henri Prade (1998) *13th European Conference on Artificial Intelligence,* John Wiley & Sons.
- Reiter E. and Dale R (2000) *Building Natural Language Generation Systems,* Cambridge University Press
- Ross, Fiona (1999) *The printed Bengali Character and its evolution.* Curzon. ISBN 0-7007-1135 X
- Savourel, Y (2001) *XML Internationalization and Localization.* SAMS
- Schmitt, David A (2000) *International Programming for Microsoft Windows.* Microsoft.
- Taylor, Dave Taylor. (1992) *Global Software: Developing Applications for the International Market,* Springer-Verlag, 1992
- UNESCO (2001) http://www.unesco.org/webworld/public_domain/kothmale.shtml.

pav hall

Unicode Consortium, The (2000) *The Unicode Standard Version 3.0*. Addison-Wesley
Winder R and Roberts G (2000) *Developing Java Software*. John Wiley & Sons
World Bank (1999). *World Development Report 1999-2000*. Oxford University Press