Linguistic Components to save Endangered Languages

Patrick Hall,

Abstract. If we are to help save endangered languages we must support them with technology so that their use is as easy as other more dominant languages. This means making software localisation as cheap as possible, amortising costs as widely as possible. The language engineering requirements to achieve this are analysed using the example of a word processor, to establish an evaluation framework which is then used to evaluate existing architectural approaches within language engineering. It is found that these approaches go part of the way, but need significant further development if they are going to be able to deliver the benefits for endangered languages that is sought.

1. Introduction - the need

It is clear that many aspects of software systems can benefit from the appropriate use of language technologies. Applications range from information searching on the Internet through to document preparation systems through to routine administrative systems. My particular interest is in software localisation, changing software so that it interacts with users in the language of their choice. Localisation is a broader problem that simply language, see for example Fernandez (1995) or Hall and Hudson (1997) or Schmitt (2000), but language localisation is the critical prerequisite and that is what I will focus on here.

Language engineering is no longer concerned with technically feasibility but with affordable cost. We are concerned with economics, is the market large enough for us to be able to recover development costs through sales of the software?

For major languages a successful product will find a very large market, and products like search engines and spell checkers can be assured of recovering the cost of their development many times over. But for most of the world's languages the speaker population is small – Nettle and Romaine (2000) note that out of the 6000 to 7000 languages of the world, only around 600 have more than 100,000 speakers. For many the number of speakers will be measured in 1000s, or even hundreds or even tens. To make things worse many of these languages may not yet be written. Even for some larger languages some other language may dominate, perhaps as the national official language, or as a unifying foreign language as in the case of English within India. Can't these small or marginalised languages be supported by technology? And if not the very smallest, how large must the population of language speakers, its 'market', be in order for it to be economically viable to develop technology for that language, to localise applications to that language? Crystal (2002) has pointed out that failure to support a language with technology may well be a contributing factor in its death. What can we as engineers, language engineers and software engineers, do about it?

If the market size for a particular single product may not be sufficient to justify the development, what then? Could we arrange our software so that we reuse the linguistic components in many applications, and thereby recover our investment over all sales of all applications that use this component? I will argue here that we can, providing that we package up the language technology into software components that can be slotted readily into a range of applications. I will view language technology as a single domain from which components should be available off-the-shelf.

I will start with presenting a well known example as a case study in localisation, showing where the linguistic components would arise. Based on this I will then describe the technical requirements for linguistic components, drawing upon experience from the software engineering community and the literature of software architectures and components. I will next survey current work within the language engineering community, to evaluate the extent to which this work has met the technical requirements spelled out earlier. Finally I will assess

what needs to be done next in order for the language and software engineering communities to contribute to saving endangered languages.

2. Example – localising a word processor

Consider a word processor, such as Microsoft Word that has been used to write this paper. However, think of a new word processor, one that perhaps has fewer facilities, but ones that are directly relevant to the task of writing documents. Imagine you are developing this word processor for world-wide markets, wanting to localise this product for all languages across the world. While in principle a word processor could interact with the user in one language while writing a document in another language (or languages), for simplicity we will assume that the same language is used for both. In today's technology this word-processor would very likely be built around XML (eg Harold 2001), but for our purpose the exact representation of the document is not critical.

The architecture of a word processor that I have in mind is shown in Figure 1. I will successively motivate the elements of this architecture

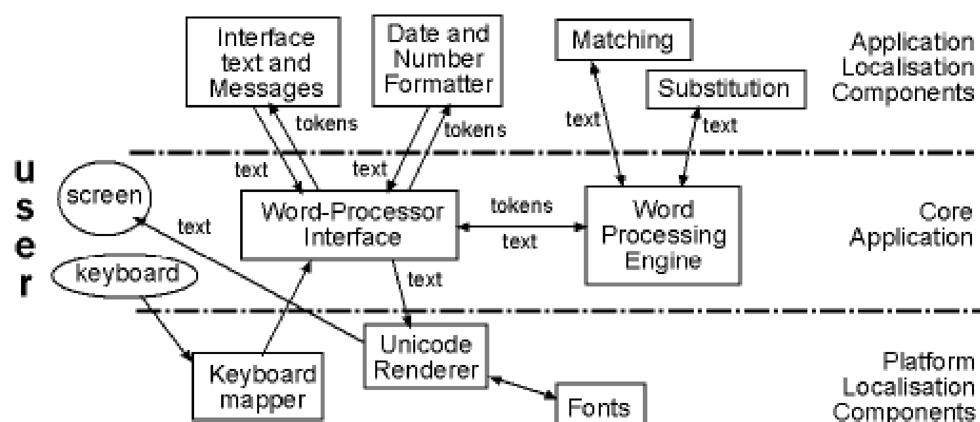

Figure 1. Internationalisation Architecture of a Word Processor

The word-processor will consist of a basic 'engine' that can store documents and their formatting, locating the position currently being worked on, and holding any temporary data such as that used in cut-and-paste operations. This engine will offer a number of facilities with which the user can add to or modify a document, or otherwise manipulate it.

These facilities would be presented to the user through a number of menus and related interface 'widgets'. The widgets would certainly contain text, and a part of the process of localisation would be to translate this text into the target language. To make this task easier for a large range of languages, all text in need of translation would be factored out into 'resource files' – all current operating systems have facilities for doing this. In addition it would be recognised that numbers and dates will be displayed differently in different locations, and the software for doing this would also be factored out into resource files. Undertaking segmentation of an application such as suggested above is known as internationalisation.

In addition some of the functions of our word processor would involve linguistic processes. For example even in something as simple as searching for a particular word, we may want to go beyond the facilities offered for English of exact matches of text fragments or of matching regular expressions; we may want to look for grammatical variants of the word sought. When we come to localise the word processor this string matcher would need to be replaced by an equivalent facility for the target language. To be able to do this easily we would need to have factored out during internationalisation the language specific elements into a component that can be simply replaced by 'unplugging' the original and replacing it by the equivalent in the target language. It is not current practice to do this, but if it were this would be done using resource files.

What is needed within the linguistic components could vary enormously from target language to target language, and we need to find a common interface to enable simple component

substitution. As part of internationalisation of our word processor we would need to abstract from all these languages the essential functions of the component interface. Within the search component we may find further commonalities between languages, from the extremes of English where given a search word we might simply look up the grammatical variants and search for these jointly using a suitable finite state recogniser, to Arabic where reduction of both search term and search text may need reduction to consonantal roots (if indeed this is feasible).

Another example where linguistic knowledge is required would be text substitution, where having searched for a word it will be replace by some other word. In English this is usually simple, but in other languages agreements of case, gender, and number may be necessary, affecting not just the replacement text but surrounding text including pronouns which may be relatively remote. Clearly the text-substitution module needs to be replaced during localisation, and in turn that would require the abstraction of a component interface for this module during internationalisation.

Returning to Figure 1, note that the core application remains the same for all target languages, and that it is only the localisation components that are replaced during localisation. Note that in the diagram we have identified some of these as application components and some as platform components, following the kind of separation that is followed in practice. The application components would usually be factored out into resource files for ease of substitution.

We can go further than the situation illustrated in Figure 1, and base the interaction with the user upon generating the interaction messages from a knowledge model of the application, as proposed for a single language by Foley et al (1989) and for multiple languages by Hall and Hudson (1997). This then requires a range of further linguistic components as shown in Figure 2.

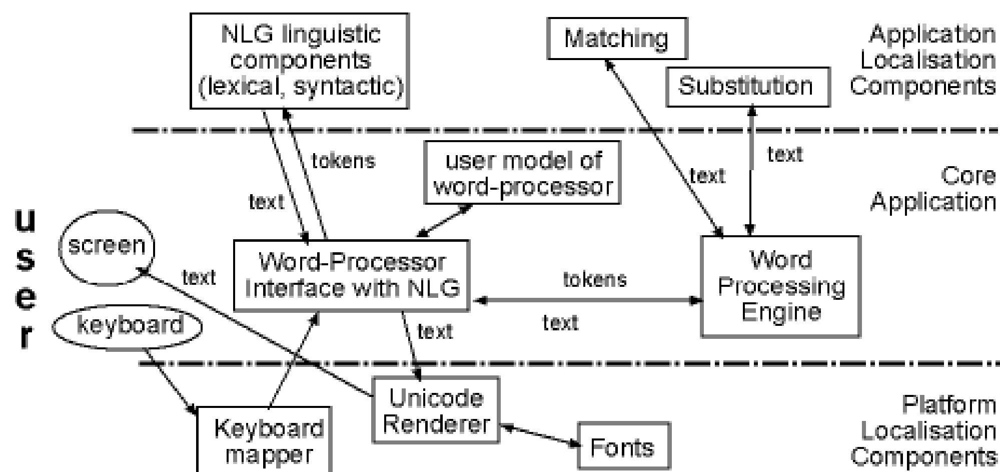

Figure 2. NLG Internationalisation Architecture of a Word Processor

3. The technical requirements for linguistic components

From the example above we can now abstract the technical requirements for a set of widely reusable linguistic components. These in turn will act as a framework for evaluating existing architectures and components within language engineering.

Firstly the writing system of the language must be encoded for the computer. Today this means being encoded in Unicode, which aims to cover all writing systems of the world. While Unicode has it shortcomings, and many omissions, it is constantly being developed and modified, and is significantly better than anything that preceded it. Early encodings simply sought to reproduce the marks on paper, if necessary decomposing letters into parts for ease of printing. We now realise that this is not necessary, and that we can focus on the alphabet and not the marks on paper, rendering the letters of the alphabet as appropriate using the increased power of computers and the versatility of printers like laser printers.

It is instructive to dwell a little on the shortcomings of Unicode, for the tendency has been to see the problem as one of encoding existing writing systems which have themselves been seen as unproblematic. Consider the letters of Arabic, which normally take a range of forms depending upon the orthographic context of the letters on either side – initial, medial, final, or isolated. All the Arabic and Brahmi derived writing system of southern Asia do this, alter the written form of the letter depending upon context. However in Arabic the conventions for writing the hamza (glottal stop) are different, it is written “on a chair” in ways which follow different rules grounded in the grammar. The result is that hamza is encoded in all its forms, presumably because the rules are either not understood by the designers of the encoding for Arabic, or the rules are too complex. Clearly to produce a correct, or even adequate, encoding can require specialist linguistic knowledge. Introducing new writing system for unwritten languages, and encoding writing systems currently unencoded, will all require linguistic sophistication. Linguists have an important role, even in something as seemingly mundane as character encodings.

The computer operating system or platform on which the software will run must be Unicode compliant. This means that the platform stores its data using the two bytes demanded by Unicode or uses the compression facility of UTF8; and the computer must include rendering engines for the writing systems concerned, typically using the Open Type technology now coming into use.

Secondly, an overall architectural framework must be defined - see for example the books by Bass, Clements and Kazman (1998) or Bosch (2000). The way the system operates in general terms must be defined as an architectural style: data flow as in Unix pipelines, or in layers of abstract machines, or data-centred around a repository, or using call-and-return mechanisms as in object-oriented systems. Key boundaries in the system must be defined: these boundaries separate out that part of the software that we are concerned with, termed a ‘domain’ by software engineers. Our domain of language engineering would be termed a ‘software domain’ by Bosch, as opposed to an ‘application domain’ such as the domain of office systems. We must define in general terms how components will be deployed, and adapted to their particular use

Thirdly the actual components in the language engineering domain need to be identified and specified. A range of typical language processing facilities must be specified by collaboration between computational linguists who will know what is computationally feasible and desirable, and software engineers who will know how linguistic components would best fit into the architectural framework. This activity is called ‘domain analysis’ by software engineers, and involves eliciting knowledge from domain experts and examining many existing systems. It is critically important in any reuse programme. All components will have their interfaces defined, both those interfaces provided to their users and those interfaces required from other components or other parts of the system. Note that a major shortcoming of object-oriented approaches is that required interfaces are only implicitly defined. The functions performed by the component will also be specified in some appropriate manner.

Fourthly these components must be readily available off-the-shelf for the major natural languages of the world. This may mean that the component interfaces should be regulated through *de jure* standards processes; this will enable them to be specified in procurement contracts and for conformance to be independently assessed via some accreditation service. Ideally the production of new components conforming to the standard should be supported by tools so that components for minor languages could be readily and cheaply produced. The set of all the interfaces to the components could be grouped together and be offered as an API (Application Programmer Interface). The components themselves should be described in some industry standard notation such as UML and realised in some widely used programming language such as Java, perhaps using the industry standard component infrastructure of Java Beans. An industry wide API and set of components for **basic** internationalisation is now entering widespread use – this is the ICU (Internationalisation Components for Unicode) supported by a consortium of major commercial suppliers of software (IBM and Others 2002); it is based on an Object oriented architectural style close to the approach used in Java (Deitsch and Czarnecki 2001). However this model does not include linguistic components and the ICU approach would need to be developed significantly further.

Fifthly, the community at large needs to commit to the use of these components. One of the barriers to the successful reuse of software has been the marked preference of many

software engineers to ignore pre-existing software and instead develop their own software (see for example Hall 1987, Tracz 1995) on the claimed grounds that it is more trustworthy and more fun. It is important that once an architecture and its APIs and components has been developed and agreed upon, it is used and conformed to, for otherwise it becomes subverted.

4. Current state of architectures in language engineering

When we worked on the Glossasoft project in 1993 and 1994, taking the language generation approach to localisation of interfaces (see above), we were very optimistic about the emergence of linguistic components that would facilitate this approach. I will use the requirements framework presented above to evaluate the extent to which linguistic components are now a reality.

The EAGLES project started about the same time as Glossasoft, and aimed, it seemed, at addressing the software reusability problem, but focused on "the interchange of tools and data among researchers and sites" (see EAGLES 1996). EAGLES joined forces with MULTITEXT and together, and then later in the follow-on project ISLE (1999), had significant successes in standardising text interchange formats for corpora and other purposes. This was important for the development of language engineering, both for researchers and for industrial developers of linguistics-based systems. The ATLAS project in the US has a similar focus and success for speech; "annotated corpora are a central component of research in human language technology". The conventions for annotation need to be described in detail, and need to be adhered to. This is an area of success for language engineering.

The RAGS (Reference Architecture for Generation Systems) project from 1998 to 2001 undertook some impressive domain analysis for NLG systems, surveying some 19 NLG systems (Paiva 1998) and identifying a set of component tasks (Cahill and Reape 1999) and then abstracting from these a 'reference' architecture reported in Cahill et al (2000a) with a set of component data elements identified in the end of project manual (RAGS 2002). The implication of some of the writing is that the intention had been to share resources on the basis of this architecture, but there is no evidence of this, and neither could there be, given the very high level of abstraction in which even specific theories of language have been abstracted away. An architectural data-centred style of the blackboard kind is defined, with shared data elements abstractly specified, but to engender reusable components it needs to go a lot further. Rather, as implied by the word 'reference', this is a framework within which existing NLG systems could be analysed, as in Cahill et al (2000b) or new systems created guided by the architecture, as in Cahill et al (2001). In order to be able to reuse and share specific linguistic resources, a much more detailed architecture would need to be developed, possibly committing to specific theories of language. And there needs to be mechanisms for enrolling people into conforming to the architecture.

GATE, a General Architecture for Text Engineering, is an impressively broader enterprise that has been in development and use for around ten years. A press release (Cunningham, Bontcheva, Maynard, and Tablan 2002) describes GATE as "an infrastructure for developing and deploying software components that process human language" that serves "companies developing applications with language processing components" as well as researchers and teachers. GATE is Unicode compliant (Maynard and Cunningham 2003) and contains a component model CREOLE based on Java Beans (Cunningham et al 2002a). GATE distinguishes two types of linguistic component, data resources (called language resources) like lexicons and processing resources like lemmatizers and translaters. And GATE has adopted a data-flow pipeline architectural style. To achieve this required a considerable analysis of the domain of language engineering, accomplished predominantly in Cunningham's PhD thesis. Beyond that GATE itself does not identify let alone specify specific linguistic components except in the context of applications, such as information extraction in ANNIE (Cunningham et al 2002 and Maynard and Cunningham 2003) where a number of specific applications have been evaluated. GATE does contain the potential for hosting a component library on the web using Java Archives, but there is no publicly catalogued component library as yet. Indeed, GATE seems to be conceived much more as "a framework and a development environment for LE" (Cunningham et al 2002b), and the

‘architecture’ describes this overall structure of tools rather than an architecture for the domain of language processing.

GATE permits ontologies developed in the Protégé knowledge-based systems development environment (Gennari et al 2002) to be integrated into GATE. The history of the development of Protégé displays an interesting parallel to GATE itself, evolving over more than a decade into a widely used environment. Protégé developed to support knowledge engineering, and focused on a set of tools to help knowledge engineers at the same time introducing a range of facilities for the reuse of inference methods, then ontologies, and then various interfacing and storage modules termed ‘plug-ins’ (see also Musen et al 2000). Implicit in Protégé and its development is a deep analysis of the knowledge engineering domain, but as with GATE there are no formal application architectures in the sense of software engineering or sharable libraries of components. Both have been more than a decade in development and evolution, it takes that timescale to achieve lasting results.

Language engineering is now being heavily influenced by the Internet. The internet provides a vehicle for undertaking language engineering, as in the use of XML (eg, Harold 2001), RDF (W3C 1999) and Jena (HP 2003), and Galaxy Communicator (Mitre 2001). The Internet also provides important applications for natural language processing, as in the semantic web, and all the above tool-sets see application to the Internet (eg. Bontcheva et al 2003).

5. Conclusions – towards the future

We started out with the hope that we could exploit libraries of reusable linguistic components in the development of software applications and their localisation. We were motivated by an interest in reducing the costs of localising software to new languages where the number of speakers was so small that they might not constitute a viable market. By deploying linguistic components over a range of applications, my hope was that more linguistic groups could be served by localised software, helping to ensure the continued existence of these languages, that these languages should not be killed off by technology.

We clearly are not yet there, though significant achievements have been made, particularly in the use of corpora with standard annotations. Looking beyond corpora, approaches like GATES are moving in the right direction, and there is potential for developing GATES to fully separate out development tools from software and application domains as has been done in mainstream software engineering, and then specialising it forward into an internationalisation and localisation development environment with suitable extensions of ICU. We can look forward to more language engineering resources becoming clearly defined such that reuse of code and data is undertaken as a matter of routine, based on libraries of components covering a wide spectrum of natural languages, viewing the whole process as a product line.

References.

ATLAS http://www.nist.gov/speech/atlas/overview.html

Bass, Len; Paul Clements and Rick Kazman (1998) *Software Architecture in Practice*. Addison Wesley.

K. Bontcheva, A. Kiryakov, H. Cunningham, B. Popov. M. Dimitrov. Semantic Web Enabled, Open Source Language Technology. *Language Technology and the Semantic Web, Workshop on NLP and XML (NLPXML-2003)* , held in conjunction with EACL 2003, Budapest, 2003.

Bosch, Jan (2000) *Design and Use of Software Achitectures. Adopting and evolving a product-line approach*. Addison-Wesley

Cahill, Lynne and Mike Reape (1999) *Component tasks in applied NLG systems.* Technical Report ITRI-99-05, Informaton Technology Research Institute, University of Brighton,. http://www.itri.brighton.ac.uk

Cahill L, C Doran, R Evans, C Mellish, D Paiva, M Reape, D Scott, N Tipper (2000a) Reinterpretation of an existing NLG system in a Generic Generation Architecture, *INLG conference*, Mitzpe Ramon, Israel, June 13-16, 2000

Cahill L, C Doran, R Evans, R Kibble, C Mellish, D Paiva, M Reape, D Scott, N Tipper (2000b) Enabling resource sharing in lanauge generation: an abstract reference architecture. *LREC conference*, Athens, May 29 - June 2, 2000.

Cahill, Lynne; John Carroll, Roger Evans, Daniel Paiva, Richard Power, Donia Scott and Kees van Deemter (2001) From RAGS to RICHES: exploiting the potential of a flexible generation architecture, *ACL01,* Toulouse, July 2001.

Cunningham,Hamish; Kalina Bontcheva, Diana Maynard, and Valentin Tablan, (2002a) GATE – a new release. Elsnews 11.1, Spring 2002

Cunningham,Hamish; Diana Maynard, Kalina Bontcheva, Valentin Tablan (2002b) GATE: an Architecture for Developemnt of Robust HLT Applications. *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*. Philadelphia, July 2002.

Cunningham,Hamish; Diana Maynard, Kalina Bontcheva, Valentin Tablan, Cristian Ursu, Marin Dimitrov (2002) *Developing Language Processing Components with GATE (a User Guide) For GATE version 2.1* (February 2003) The University of Sheffield 2001-2002, http://gate.ac.uk/

Crystal, David (1997) *English as a Global Language*. Cambridge University Press.

Crystal, David(2002) *Language Death* Cambridge University Press; ISBN: 0521012716

Deitsch, Andrew and David Czarnecki (2001) *Java Internationalisation*. O'Reilly.

EAGLES (1996) *Linguistic Software Reusability*, Document LSD1 version 1.0 28 April 1996. http://www.lpl.univ-aix.fr/projects/multext/LSD/LSD1.main.html

Fernandez, Tony (1995) *Global Interface Design. A guide to designing international user interface*. Academic Press.

Foley, James; Won Chui Kim, Srjdan Kovacevic and Kevin Murray (1989) Defining Interfaces at a High Level of Abstraction. *IEEE Software* January 1989 pp 25-32

Gennari J, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, S. W. Tu (2002) *The Evolution of Protégé: An Environment for Knowledge-Based Systems Development.* SMI Technical Report Number: SMI-2002-0943

Hall P.A.V. (1987) SOFTWARE COMPONENTS reuse - getting more out of your code, *Information and Software Technology*, Butterworths, Jan/Feb 1987

Hall P.A.V. and R.Hudson (1997) *Software without Frontiers. A multi-platform, multi-cultural, multi-nation approach*. John Wiley and Sons.

Harold, Elliotte R. (2001) *XML Bible*. 2nd Edition. Hungry Minds

HP (2003) *Jena Semantic Web Toolkit – Data Sheet*. http://www.hpl.hp.com/cgi-bin/pf.cgi

IBM and others (2002) International Components for Unicode (ICU) *<http://oss.software.ibm.com/icu/userguide>*

ISLE (1999) D14.2 *Final Report,* Feb 2003

Maynard, Diana; and Hamish Cunningham (2003) Multilingual adaptations of ANNIE, a reusable informaton extractin tool. *Procedings of EACL 2003*, Budpest Hungary April 12-17. pp 219-222.

Mitre (2001) *Galaxy Communicator Tutorial: Getting Started*. http://communicator.sourceforge.net/sites/MITRE/distributions/GalaxyCommunicator/docs/manual/index.html#Tutorial

Musen M A, R. W. Fergerson, W. E. Grosso, N. F. Noy, M. Crubezy, & J. H. Gennari. (2000) Component-Based Support for Building Knowledge-Acquisition Systems. *Conference on Intelligent Information Processing (IIP 2000)* of the International Federation for Information Processing World Computer Congress (WCC 2000), Beijing, . 2000.

Nettle, David and Suzanne Romaine (2000) *Vanishing Voices. The Extinction of the World's Language*. Oxford University Press.

Paiva, Daniel S (1998) *A Survey of Applied Natural Language Generation Systems*. Technical Report ITRI-98-03 , Informaton Technology Research Institute, University of Brighton,. http://www.itri.brighton.ac.uk

Schmitt, David A. (2000) *International Programming for Microsoft Windows*. Microsoft.

Szyperski, Clemens (1998) *Component Software. Beyond Object-Oriented Programming*. Addison-Wesley

Tracz, Will (1995) *Confessions of a Used Program Salesman. Institutionalizing Software Reuse*. AddisonWesley

Unicode Consortium (2003) *The Unicode Standard, Version 4.0* Reading, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1)

w3c (1999) *Resource Description Framework (RDF) Model and Syntax Specificaton*. W3C Recommendation 22 February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222