The State of Software Technologies Affecting NR Solutions

NRSI Field Advisory Board, February 2000 (Update notes added October 2000)

Peter Constable,
SIL IPub/Non-Roman Script Initiative (NRSI)

1. Introduction

The state of technologies that impinge on support for non-Roman scripts has changed considerably over the past three years. This paper attempts to highlight those changes.

This paper was originally prepared primarily for the benefit of the Non-Roman Script Initiative (NRSI) and the NRSI Field Advisory Board as they were nearing the end of an initial three-year planning period, and working on a plan for the coming three years. I have made some revisions to make it more suited to a wider audience, and also to bring it up to date with further developments in the past eight months.[1]

There have been a number of exciting developments in the past three years. Some of the issues we face today have been with us all along, some have been around but have changed, and others are completely new. As we evaluate where we need to go in providing solutions for working with multilingual and multi-script text, I thought it would be helpful to get a quick picture of the overall landscape so far—where we've been and where we're at.

So hop on board on this (still long but relatively) quick tour. We'll start by looking at the state of the art as we saw it three years ago when we worked on our first three year plan. Then we'll move on to look at the scene we face today. Our tour won't be stopping along the way to do any detailed sight-seeing, but we will cover a lot of interesting ground.

2. The State of the NR Art in Early 1997

We prepared our first three year plan in early 1997. Here is a description of the state of technology we were dealing with then, and of how we were making use of it.

2.1 Encoding of Text Data

All of our text data was encoded using 8-bit encodings. In some cases, this may have involved multi-byte encodings, but most cases were single byte.

We were aware of Unicode, but it was still only a flicker on the horizon. Quoting Bob Hallissy from a presentation at NRTC: "[The] standardization effort is not complete enough to answer all of [the] encoding

[1] Additions that represent key changes in the current "state of the art" are indicated by italic type, and by the introductory text, "*Oct. 2000 update*". These are not the only revisions that were made, though they are the most significant.

issues we as an organization face... Windows users are impacted by Unicode, and it causes various difficulties with font compatibility on non-US versions of the OS."

All of the software we used was built on an 8-bit & codepage model. In nearly every situation, we used a standard Western codepage setting for the OS or other commercial software that might be involved, but we redefined ("hacked", frankly) the definition to provide other "special" characters.

2.2 OS & Commercial Software

Microsoft software was essentially based on a localisation model in which support was provided for English plus one other language as needed for a particular market. The majority of SIL users were using US versions of software. Thus, in most cases, we were working with modifications of Windows codepage 1252.

Some localised versions of Windows provided support for NR scripts, but in these cases the support was always hard wired and buried relatively deeply in the OS. For example, Thai Windows 3.1 had a special version of the GDI component, and Thai fonts needed to have a special, undocumented flag set for Thai rendering rules to be applied. These implementations were far more often a hindrance rather than a help.

In contrast to Microsoft's localisation approach, Apple software was based on a global approach and was capable of supporting multiple languages using WorldScript technologies, though only certain apps were designed to use it, and some did so imperfectly. The Mac also provided superior rendering support using TrueType GX, though even fewer applications existed that supported that.

In certain respects, the Mac OS infrastructure was not as built up as the Windows infrastructure (though in ways that didn't affect us a lot at the time). For example, codepages have always been a Windows thing, and TrueType fonts on Windows used a Unicode-based cmap table, while fonts on the Mac used an 8-bit cmap. Windows 95 introduced some support for Unicode, which only came later on the Mac, but this wasn't particularly affecting us at the time. Windows 95 also introduced some new "national language support" infrastructure elements, and these were used in Office 95. But again, these did not seem that significant to us at the time, and none of the applications we were familiar with on the Mac OS were using anything similar.

As far as SIL users were concerned, we knew that a number were using Macs, some specifically because of NR-related needs, but we also knew that there were several fields for which the Mac was, for various reasons, not considered an option as a general solution to NR-related needs.

Most SIL users were using Win 95, though many were still using Win 3.1. Window NT was being used only rarely if at all, though we knew the number of users might increase in the future.

We were keenly aware of the dependence that SIL users have on commercial software, in particular for publishing (linguistic articles, primers and literacy materials, dictionaries, translated texts—though the major typesetting jobs are often handled by publishing specialists using special apps rather than by OWLs using business apps). Part of the major dilemma for us was that, on the one hand, we couldn't control the level of NR behaviour in commercial apps, but that, on the other hand, we didn't particularly want to get involved in developing our own word-processing and DTP software. In early 1997, though, we had just come to the point of deciding that, even if we develop means to provide NR support that only works in SIL apps, then that's better than doing nothing and would be worth the effort.

(For more comments on publishing, see §2.7 below.)

2.3 Rendering and Font Technologies

For rendering on Windows, "dumb" fonts (that is, fonts rendered without any complex script support, using a one-to-one mapping from characters to glyphs) were it. This meant an effective limitation of at most 221 glyphs that were accessible from a font, and that text had to be encoded in terms of presentation forms (glyphs), with input methods handling character-to-presentation form mappings.

To facilitate creation of fonts for such situations, the Encore package with the TypeCaster tool was available. At the time, work was being done on version 3, which added the catalogue and visual editor. These new tools could only be used with the Encore library fonts, however, although the compiler could be used with both Encore and non-Encore fonts.

Within SIL software, most users were working with the above limitations. A new capability had *just* been added to LinguaLinks software: the ability to create “transduced fonts”, for which text would be passed through a CC table before being drawn on a display. This had some very obvious limitations (for instance, inability to handle range selections correctly), though it could have some small benefits for certain users. (I don’t know that this capability has ever been seriously utilised.)

Tim Erickson was nearly finished with development of the first version of SDF, support for it had been added to LinguaLinks, and Tim was working closely with the Shoebox development team to get support incorporated into that product. At CTC in November 1996, we had met together with Tim and with reps from the various NR entities who were present to discuss SDF’s potential as a solution. During that meeting, Tim indicated that he didn’t expect SDF to be able to handle Indic reordering or class-based substitutions. SDF was also based on 8-bit technologies and therefore not able to handle large character sets or even some complex, small-character-set scripts (due to limitations on the number of glyphs that could be used), including some potential Arabic implementations. For these reasons, it did not seem that we could count SDF as a truly general purpose solution, and so it wasn’t a major factor in our three-year plans. (We had considered the possibility of multiple complex rendering solutions aimed at different script families, but for various reasons, a single, general purpose solution was deemed to be far more preferable.)

We were aware of rumours and “promises” of other technologies from commercial developers for Windows, but none actually looked particularly promising to us. There were hints that Apple might provide GX typography on Windows NT, but this didn’t look that likely to happen. Microsoft was working on TrueType Open (TTO) and had promised to provide a TrueType Open Services Library (TTOSL) for developers (only recently delivered—three years later!). TrueType Open was being used by MS for some solutions, e.g. Arabic Windows 95, but most of this was hardwired deep in the OS. There wasn’t even a specification available for TTOSL, let alone software. Furthermore, MS’s documented philosophy on rendering support was clearly not going in the direction we needed. From the TTO 1.0 spec:

> [TTO tables] do not try to encode information that remains constant within the conventions of a particular language or the typography of a particular script. Such information... belongs in the text-processing application for that language, not in the font.

The implication of this was that rendering behaviour for complex scripts was the responsibility of application developers, which would be entirely unsatisfactory for us for several reasons. For instance, it means that each application is responsible for handling complex script behaviours, with the potential that different applications would do this in incompatible ways and would each rely on proprietary fonts.

In other words, the outlook for Windows was very bleak from our perspective.

On the Mac side, WorldScript and GX provided very capable technologies that were being used successfully in several projects. The main limitations were that only a handful of applications used those technologies, and the Mac was not considered an option as a general solution for some fields. Jonathan Kew had built tools for creating WorldScript and GX resources, though these weren’t intended for use by OWLs or even typical field support personnel.

For very-large-character-set (VLCS) fonts needed in East Asia, we knew that there were issues in providing fonts having to do with the large numbers of glyphs (we were having problems getting Fontographer to work with tens of thousands of glyphs) and the use of double-byte character set encodings and codepages. We in the NRSI didn’t have much experience in dealing with Far East problems, however.

Some scripts have characters involving fine detail that can get lost at small sizes. This was particularly a concern on low resolution devices, and felt to be a particular problem for Arabic script. Two solutions were available to us: font hinting, and inclusion of bitmaps in TrueType files. Both were very labour intensive, however, and required special training, tools and skills. We really didn’t know how best to deal with this.

The tools that we depended on for building fonts were primarily Fontographer 4.1, TTAdopter, TypeCaster, and Jonathan's WorldScript and GX tools.

2.4 Conversion Tools

Our standard tool for data conversion was the Consistent Changes (CC) program. A few people were starting to take an interest in Perl, and we sensed that new approaches were needed in the future. But CC was sufficient (even if not ideal) for most of our perceived needs. Something to note in particular: CC was designed for 8-bit text and not 16-bit text, but that limitation was not a concern for us at the time.

2.5 Input

Our standard tool was Keyman 3.2, which had recently been completed by Wes Cleveland. (Version 3.2 provided the ability to work on Win 95.) This worked well for the 8-bit, hacked-code page 1252, presentation-form encodings of text that were the norm for most of our users at the time.

For East Asia, we knew that special means of inputting text were needed, but we (NRSI-Dallas) really didn't know much about the extent of the need, or about the standard technologies used for this, just as we knew fairly little about Far East computing issues in general. If anything, we felt the need for a Far East specialist to join our ranks.

On the Mac side, WorldScript provided means for defining keyboard layouts, and where more power was needed, Jonathan Kew had developed SILKey. Like Keyman 3.2, this was designed for 8-bit text.

2.6 Text Analysis

There are other text analysis operations of particular interest in connection with NR scripts: searching, sorting, line breaking, concatenation, case mapping. We hadn't given much thought to these operations in particular, though, assuming that the capabilities existing in apps were probably adequate for most cases. These weren't show stoppers, such as simply having no way to display Ethiopic in Shoebox, and there were workarounds for some of the more important problem cases. E.g. Shoebox provided the ability to use CC to generate a sort key, which is needed for Thai sorting to handle "reordering" behaviour, and Thai line breaking could be handled by inserting a zero-width word/syllable separator. We had just discovered a new problem area, though, also related to Thai: LinguaLinks failed to maintain the original word breaking of imported texts and would concatenate word forms into a text by adding a space between all word forms. All in all, though, these concerns were low in our priorities.

2.7 Large Scale Publishing

At the time, large typesetting jobs within SIL were usually being done in Ventura or TeX. Jonathan Kew had developed TeXgX, which solved complex rendering problems in a sophisticated DTP app, and was using it successfully in East Eurasia Group (EEG). It required that GX fonts be developed, which had only been done in a few cases, but it was a solution with considerable potential. It assumed 8-bit text.

Lorna Priest had typeset a number of book-length Ethiopic script documents using Ventura, though it involved a very complex system based on CC and worked on multi-byte-encoded texts that had been prepared using the ED program in DOS. In Ventura, the large character set was handled using multiple TrueType fonts, and any changes to the text had to take place in a separate text editor such as ED.

A DOS-based system involving Ventura, CC and bitmap fonts was also being used for Devanagari.

The Tai Dam project (NAB) was using a combination of GX fonts and a GX-capable DTP application (Ready, Set, Go GX) on the Macintosh to do some publishing, but found it cumbersome and problematic (due to limitations in DTP-related features of the application, not due to the rendering technology).

Beyond that, the only attempts at publishing were being done using programs such as Word with the typical hacks to handles scripts like Thai.

Dennis Drescher had become particularly interested in the publishing process, and in ways in which the technology used for preparing text for typesetting could be greatly improved. This led to his R&D work called "Project Kokou". This would be beneficial to NR publishing, though it wasn't specific to NR scripts.

2.8 Formalisation of NR-related Requirements for Language Software

In early 1997, we certainly had some ideas of what was required of language software in order to provide adequate support for NR scripts, but we had not made any attempt to collect and organise any of those ideas into a formal statement.

2.9 Software Localisation

In 1997, we had not particularly thought about localisation of SIL software. In a few instances, French or Spanish versions of SIL software had been created using an English-first/localised adaptation development model (build an English version, then make a second, separate version for another language by going through the code making changes wherever needed to suit the needs of the other language). This methodology is potentially very inefficient and prone to problems, and is considered to be bad methodology by those with the most experience in software globalisation in the industry today.

2.10 Language Identification

Our standard practice with SIL data was to use SIL *Ethnologue* three-letter codes to identify the language associated with data. This use of *Ethnologue* codes was built into LinguaLinks, but no other SIL software. No industry standards for language identifiers were well-established in areas that affected us. Overall, this was not an area we were concerned with.

3. The State of the Art in Early 2000 (updated October 2000)

In the three years since the snapshot described above, a number of changes in technology have occurred. In this section, we will get a snapshot of the current state of the art.

3.1 Encoding

Most of our users and many apps (in particular currently shipping SIL apps) are still working with 8-bit text. The writing is plainly on the walls, however: Unicode is here and is rapidly taking over. Already, the business apps that most SIL users are using, the MS Office suite, are built around Unicode. New SIL software that is being prepared is being designed to work with Unicode.

Version 3.0 of Unicode was published early this year, and a minor update (version 3.01) has since been released. With version 3, Unicode now supports scripts that account for a major portion of the remaining languages we expect to work in. There are still some scripts it doesn't support, however; e.g. Vai, Tifinagh, Dehong Tai, New Tai Lue, Silhoti. There are also occasional gaps in the scripts it supports—characters required for particular languages; e.g. GHE WITH RIGHT DESCENDER (required for Yupik and at least one other language). There is a good level of confidence that we can get these added in future versions of Unicode, however, and in the mean time we can utilise the Private Use Areas provided by Unicode.

Microsoft and Apple are both solidly behind Unicode. Microsoft is a little ahead, having introduced Unicode support in Windows 95 and Windows NT 4, though Apple has added full Unicode support in the Mac OS (beginning with version 8.6), and is strongly encouraging developers to move in that direction. Most major industry players are quickly moving to get on the Unicode train, driven in part by the fact that Unicode is the default encoding for some key Internet technologies, such as XML.

We are in a transitional period in which we still need to deal with the problems that 8-bit encodings and codepages caused, but our new efforts our focused on Unicode.

Some initial decisions were made at CTC in 1998 regarding how we as an organisation would manage our use of the Unicode Private Use Areas. We are at the point now where we are needing to begin implementing mechanisms for making specific decisions about PUA characters and for documenting our use of the PUA. This is high on our list of priorities, but is still waiting to be done.

3.2 OS and Commercial Software

A paradigmatic change has occurred in Microsoft software: they have entirely abandoned the multiple-localised code base model, and replaced it with a single, global code base model. Windows 2000 and Office 2000 are built as single versions of the code that is capable of handling text for any of the languages of the world that they have targeted.[2] They still package localised versions of software, but these are distinguished only by the non-core items (e.g. proofing tools and templates). This is of considerable benefit to us: it means that we only need to research and understand one version of their software, it makes a higher general level of NR-related capability available to all users, and it makes it possible to create documents that contain multi-lingual text where significantly different categories of script are involved (e.g. Arabic and Chinese).

Most SIL users are currently using Windows 9x, which has some limitations in comparison with Windows NT and Windows 2000 in relation to Unicode support. The most important limitation has to do with input methods (see §3.5); another area of limitation is with file naming.

Oct. 2000 Update: Windows Me, the successor to Windows 98, has now shipped, and is starting to appear on OEM systems. In relation to multilingual and multi-script support, it is not significantly different from Windows 98.

Earlier this year, we approached some key contacts at Microsoft regarding the inherent limitations with keyboard input in Window 9x, and some changes have been made. See §3.5 for details.

Windows 2000 is starting to ship, and its global design means that this may be an important platform for our use. It is more suited to use on portable computers than NT4 (making it comparable to Win9x), and much more stable than NT4 (or Win9x). (It is not well suited for home use where games and educational software may be important.) The recommended minimum requirement for RAM is 64MB, though 128MB significantly improves performance (according to PC Magazine). The OS also requires about 650MB of hard disk space (not a problem given the current capacities of hard drives). Even if large corporations are slow to adopt this OS, it will probably still have a major impact. Since it supports Plug-n-Play and uses the same hardware drivers as Win9x, it should be very well supported by hardware manufacturers.

Oct. 2000 update: while it is clearly part of the Windows NT product line and has a networking administration layer that can be daunting to untrained users, Windows 2000 looks very promising for use within SIL. The reports I have heard from early adopters within SIL who have used Windows 2000 Professional on laptops have been generally positive. It is without question the current best OS from Microsoft in terms of support for non-Roman scripts. There are a variety of factors users need to consider in selecting which OS to use, however, particularly with regard to training and support. This is a different operating system from Windows 98, for example, and a user should not consider switching from that to Windows 2000 unless they are prepared to learn to use a different operating system.

For such reasons, we cannot make a blanket recommendation that entities or individual users switch to Windows 2000. It should certainly be given careful consideration by field entities, however. For example, if an

[2] This statement is not actually 100% true: the code for supporting South- and Southeast-Asian languages in Office 2000 was not quite ready on time, and a separate binary was used for localized versions of Office for that region. This is a marked difference from previous versions, however, and the orientation toward a single global binary is now well established within both the applications and the systems divisions of Microsoft.

entity has a number of members from countries such as Korea and Japan, computer support personnel may face difficult challenges supporting Korean and Japanese versions of Windows 98 in addition to the North American/European version. Standardising on Windows 2000 could simplify the work of support personnel in this regard. Windows 2000 is by no means a panacea, however: it may not be accessible to some of the non-SIL colleagues we work with, and its multilingual support is very often still not adequate for the needs of minority languages.

Window 9x and Windows NT/2000 provide an infrastructure for multilingual support— "national language support" (NLS)—that was introduced in Windows 95. It has had some benefits, but it also has some important limitations. In particular, NLS provides a system for identifying languages that supports a maximum of 512 primary language distinctions. (Each primary language can include 32 distinct sub-languages; e.g. British vs. Australian English.) This is a serious limitation given the scope of languages that interest us.

Microsoft are currently working on a new infrastructure—NLS+—that addresses some of the limitations inherent in NLS. I know that, for language identification, they plan to use the three-letter tags of the ISO 639-2 standard. This doesn't provide an immediate solution to our concerns due to the limited definitions in ISO 639-2. It may provide the kind of extensibility we are looking for, however. *Oct. 2000 update: NLS+ is but one element in a new software framework that Microsoft has adopted known as ".Net" (pronounced "dot net"). I have not had sufficient opportunity to evaluate the .Net developments to comment on the overall potential impact for our work.*

My contact at Microsoft who is responsible for work on NLS+ has given some indication of a desire to see tools available to allow implementers to provide additional multilingual-related capabilities in Windows, the very kind of extensibility we have long desired. (These wouldn't necessarily extend to all aspects of script-related capability, however, particularly rendering.)

Oct. 2000 update: in August of this year, Martin Hosken, Marc Durdin and I spent several days at Redmond interacting with some of their engineers and development managers. We received further indications of possible openness toward adding capabilities for extensibility of multilingual features in future versions of Windows, including in relation to rendering. No particular commitments have been made, however, and I will refrain from elaborating on areas that at this point are no more than mere speculation.

Microsoft still builds software for the particular markets they have targeted, and providing capability using their software for additional languages continues to be a challenge. As I was recently told by one Microsoft contact, for instance, "Word97 (and even Word2000) were not designed to support any languages we didn't test, and we didn't test Yi." For the present, however, the climate at Microsoft seems somewhat more amenable to our multilingual needs than it has in the past, and we have some contacts that have provided valuable help and influence, even if only in limited ways.

Oct. 2000 update: we have learned that the systems division has the intention of eventually supporting within Windows at least every script in Unicode version 2.1. (Some, but not all, scripts that are new in Unicode 3.0 would also be supported.) This would include eventual support for all major scripts of India, and also Lao.

Meanwhile, Apple has provided a third and mature generation of technologies for multilingual and script-related support—ATSUI, Text Services Manager and other related text technologies. Applications that support their technologies are still few, however, though they are working to make it easier for application developers to incorporate support for these technologies into their products. We continue to have good interactions with people at Apple, though the amount of interaction may be less than it has been at times in the past. *(Oct. 2000: they recently elicited input from us in writing a specification for certain enhancements in their rendering technology, though.)*

If anything, the SIL landscape for Mac users has grown more difficult, particularly with certain decisions regarding software development made last year. There is a current effort being made to keep Mac software development within SIL alive, but it is too early to tell to what extent this will succeed.

As far as I know, the field entity perspective regarding whether the Mac can be a viable platform for language projects is largely unchanged from three years ago. One entity, ETG, made a decision to consider the Mac as a component in the solution to their script-related needs. Beyond that, some entities strongly endorse the Mac (e.g. EEG), other entities favour Windows but don't actively discourage Mac use by field members, and in some entities the Mac is not considered a seriously viable option for reasons including availability of support, lack of local presence, and the overwhelming adoption of Windows by non-SIL partners.

Java and Linux have been touted as potential driving forces in the future. Within the world of SIL, however, we have not yet come to a point at which we consider these to be likely to have any imminent impact on language software or on our language projects. Java provides some useful non-Roman capabilities, but it does not yet support all that we need. There are additional reasons why it was not considered as an option for FieldWorks development at this time.

Oct. 2000 update: there continues to be growing interest in Linux within industry in general, and considerable work is being done, much of it through open source collaboration, to build capable software that runs on the Linux platform. Included in this are efforts to improve multilingual capabilities on Linux. We need someone to do further research into Linux and to monitor its potential for benefit in our work.

As in the past, SIL projects continue to rely heavily on commercial software especially for publishing. Other than the general improvement for multilingual and script-related support in applications like Word, little has changed in this regard. *Oct. 2000 update: there have been some developments that affect large-scale publishing. These are discussed in §3.7.*

3.3 Rendering and Font Technologies

Apple's TrueType GX technology continues to be with us essentially unchanged except for the name: *Apple Advanced Typography* (AAT). The QuickDraw GX interface that application developers had to work with has been abandoned and replaced by *Apple Type Services for Unicode Imaging* (ATSUI), but no capability has been lost along the way. (QuickDraw GX support was dropped with OS 8.5. It is possible for GX apps to continue to work through version 9.0, though this requires a special download from Apple's web site, and this option may be dropped at any time.) There are ways in which AAT could be enhanced, but this has been low among the priorities for Apple management, so little new has appeared in the past three years.

ATSUI has, unfortunately, floundered. It has not been significantly adopted by developers largely because it was costly for many: the interfaces were different from both QuickDraw GX, Worldscript, and the pre-Carbon plain-vanilla text-rendering interfaces, and because these interfaces were not supported on any OS prior to 8.5. (This may have since changed, but too late for many vendors.) The small momentum that existed for GX development has taken a hit, and appears not to really have recovered.

To make up for the lack of developer interest in ATSUI, Apple has recently introduced another alternative for developers: MLTE ("Multi-Lingual Text Edit"). This is a text edit control built on ATSUI that is comparable to Microsoft's Rich Edit control. It supports the same level of rendering support provided in ATSUI, including selection of font features. They are also working on adding lower-level interfaces in ATSUI, which will be of interest to some developers for whom a fine level of control is important. (MS Word is a case where this level of support has been wanted.)

When Apple acquired Next, they added a second set of interfaces and services for developers to build on: NextStep (also known as, "Rhapsody", "Yellow Box", "Cocoa"). This provided Unicode-based text handling, with some support for smart rendering, but separate from (and not as complete as) ATSUI. In particular, there is no support for right-to-left text. The International and Text group at Apple has expressed their wish to see international text facilities unified, with Cocoa text becoming a client of ATSUI (or the underlying lower-level services), but the Cocoa team does not appear to see it as a high priority, and as far as we know there is no progress yet towards this.

Since the AAT font format used for ATSUI is identical to TrueType GX, Jonathan Kew's GX font tools continue to work. They have not been further developed in the past three years.

Oct. 2000 update: one of the few areas in which TrueType GX/AAT has lacked is that it has not supported positioning by means of attachment points. This was something that we requested when Apple was first designing the GX system, but for some reason they chose not to. They are now revisiting this, and it appears that attachment point capability will be added to AAT in a future version of the Mac OS.

In the past three years, Microsoft entered into a co-operative effort with Adobe, the result of which is OpenType. OpenType supercedes TrueType Open, and also supports Adobe Type 1 (Postscript) font outlines. OpenType has become a key technology for Microsoft, and is used in all of their current versions of software. They have recently made available to developers the OpenType Services Library (was promised more than three years ago). This provides developers with means of accessing rendering information included in the OpenType tables of a font. These services conform, however, to the design philosophy of TTO mentioned above in §2.3 that placed the burden for understanding script behaviour on the application developer. It is unclear at this time whether or not this can be useful to us.

As Microsoft continued developing applications for global markets, they were encountering some of the problems with their design philosophy. This has brought about a significant shift for Microsoft: their software now relies on a centralised component, Uniscribe, to shield applications from the complexities of rendering various scripts. This has been a positive step which has made it possible to create single, global versions of Office and Internet Explorer. It also allows them for the first time to provide rendering support for complex scripts in those apps without requiring the corresponding localised version of the OS. So, for example, it is now possible to render Arabic text in Office 2000 running on US Windows 9x.

While Microsoft's statements about their plans for Uniscribe are still vague, it seems fairly clear that Uniscribe has become a core part, along with OpenType, of the smart font rendering mechanism for MS application developers. *Oct. 2000 update: beginning with Windows 2000, Uniscribe and OpenType are used by the standard Win32 interfaces that applications use for displaying text, and so some existing applications can now get some degree of complex-script rendering support for free.*

At the same time, the OpenType/Uniscribe architecture still has some limitations. In particular, we still face a situation where script-related rendering behaviour is hard wired into an OS component. In the past, it was in localised versions of GDI.DLL; now it's in the single, global version of Uniscribe. There are some other limitations in Microsoft's current approach to rendering that affect us, but this is the most serious, and Microsoft seems unlikely to change in this regard. OpenType and Uniscribe hold a lot of potential to benefit us, but serious questions remain, with little assurance that our needs can be adequately met.

Oct. 2000 update: in recent months, there have been some significant indications of a change in perspective from Microsoft contacts on this. They are now very interested in creating an extension mechanism that will make it easier to add new rendering behaviours into Uniscribe. It is still unclear to us how far this may go, however, and whether it will be something that we can use to meet the rendering needs of minority languages.

One additional note on Microsoft's and Adobe's co-operation on OpenType: one of the result's of Adobe's move to OpenType is that multiple master font technology will no longer be supported. It is unclear to me at present whether it will be possible to continue using existing multiple master instances on new software.

Oct. 2000 update: Apple have indicated their intention to support OpenType natively in future versions of the Mac OS beginning with OS X. This will be in addition to AAT/ATSUI. An initial step has already been taken in that Mac OS 9 now supports Windows-style data fork TrueType fonts. Since OpenType depends upon client software such as Uniscribe that has knowledge of script behaviours, something will be needed to fill this gap in order for support of OpenType to entail complete support for rendering complex scripts. Apple has indicated that they intend to address this problem, but have not given any indications as to how.

The transduced fonts introduced in LinguaLinks 1.5 remain in that product, but have never been seriously utilised, and are of little future benefit. In the mean time, Tim Erickson has provided SDF, which has been supported by both LinguaLinks and Shoebox, and which has been of considerable benefit, especially for teams in Eurasia. In early planning for FieldWorks, it was anticipated that support for SDF would likely be provided in FieldWorks. Given that FieldWorks is built on Unicode, however, and that SDF is designed to work with 8-bit text, there are some technical issues that would need to be resolved to make this possible.

SDF has undergone some moderate enhancements since it first appeared, the most significant of which is that it now supports the ability to access thousands of glyphs. It is still limited to working with 8-bit text, however, and this is unlikely to change. The implication of this is that SDF is likely to remain an interim product only. It will likely continue in use for several more years together with Shoebox and possibly ScriptPad (which is not yet completed). There has been some thought to incorporating SDF into Paratext, though I don't know how certain this is to happen.

At the time of writing, substantial progress has been made in realising the number one priority goal of our initial three-year plan: WinRend, now dubbed "Graphite". At the current rate of progress, there is a high degree of confidence that Graphite support will be included in the first shipping FieldWorks products, scheduled for the fall of 2000. Graphite still does not provide solutions where commercial software is involved, but we have a high degree of confidence that our needs for rendering support in SIL software can be adequately met.

Oct. 2000 update: a beta version of Graphite has been in use by various testers for several weeks now. It is being tested together with a pre-release version of WorldPad, a simple text editor that uses the Graphite rendering engine. While further optimisation and fixes are needed, the software has been performing quite well. One of the initial design features (hinted attachment points) had to be dropped from this release due to a bug in Windows, but this does not represent a serious problem at this stage, and we have been told by contacts at Microsoft that the bug is being fixed.

The core elements of the TrueType font format, on which all of our solutions have been based, remains unchanged except for a few minor enhancements. The general font-development tools we depend on have also not seen much change. In particular, we still use Fontographer 4.1, and it has become increasingly apparent to us that Macromedia is unlikely to make any further enhancements to Fontographer. Among type designers, RoboFog and especially FontLab have taken over in popularity.

TypeCaster, TTAdopter and Jonathan's WorldScript and GX tools continue to be key parts of our tools box. Some significant additions have been provided by Martin Hosken, based on a set of Perl libraries he developed. Microsoft has also just recently provided a beta of their Visual OpenType Layout Tool (VOLT), which can be used for adding OpenType tables to a font. *Oct. 2000 update: VOLT 1.0 has now been released.*

Three years ago, we anticipated the need to extend the functionality of TypeCaster to allow the editor and catalogue to work with non-Encore fonts. There has not been opportunity to begin work on this, and we should probably re-evaluate our needs in light of the current technology situation. We anticipate that there may still be a need for enhancement to TypeCaster, but the nature of the needs may have changed. In particular, Unicode and Graphite were not factors three years ago but are crucial factors today.

Oct. 2000 update: While the TypeCaster compiler can be used with non-Encore fonts, there is a limitation in relation to how glyphs in a source font can be referenced in that character references are limited to the decimal range of 0–9999 (U+0000 to U+270F). This represents another way in which it may be useful to extend TypeCaster.

Needs in relation to hinting that were felt three years ago remain. Microsoft has provided a powerful tool for hinting, Visual TrueType, but we haven't had anyone available who could develop the expertise to use this effectively to hint our fonts. Microsoft has another technology under development, ClearType, which is scheduled to begin appearing very soon (software for testing purposes may be available immediately). ClearType reportedly improves legibility on low resolution displays considerably, especially on LCD displays. (It is as yet unclear whether anything in this technology might limit its effectiveness to only Roman script.) In addition, there are industry projections that display technology may see some significant advances in the next few years with the result that our displays will jump from a typical 96 dpi to around 200 dpi. These two technologies may significantly address existing concerns about legibility of type at small sizes. *Oct. 2000 update: ClearType has since been released as part of Microsoft Reader—an application for reading electronic books that is freely available from Microsoft. ClearType support will be appearing with future versions of other Microsoft products, such as Office and Windows, but we do not know when. We have not had adequate opportunity to evaluate what benefit ClearType may offer us as an alternative to careful hinting of fonts.*

Another key area of font technology development in the past three years has been in relation to the Internet: Microsoft and Netscape are each promoting embeddable font technologies that permit an author to publish a document on the web and to be sure that the reader will be able to read the document with the same type without the reader having to download and install fonts. One major hindrance is that Microsoft and Netscape are promoting competing technologies, neither of which has come out ahead. Another is that neither technology currently allows users to use these web fonts on pages without significant effort and control over the system hosting the individual pages.

3.4 Conversion Tools

There have been only a few changes here. CC continues to be the main tool, and continues to be designed for 8-bit text. It has been extended in a way that means it *can* be used for 16-bit text, but it is not ideally suited for that purpose.

Ken and Darrell Zook have built a conversion module that is likely to be used in some manner within FieldWorks, but it is not ready as an end-user tool. In particular, there is no user interface available for invoking it; it is only a COM object that can be called upon by other software.

Perl has a growing user base within SIL, though it is still far from ubiquitous.

We need a conversion tool that supports Unicode and 8-bit-to-Unicode (and vice versa) conversions and is accessible for the average user.

Oct. 2000 update: The conversion tool that Ken and Darrell Zook created was designed primarily for certain needs they faced in the development of FieldWorks. It was not clear to us that this tool would be completely adequate for the full range of conversion needs that we faced. This was a particular concern in relation to legacy encodings that directly encode presentation forms (used in many SIL projects),such as ligatures and contextual variants.

Similarly, work was started by the Unicode Technical Committee to create a standard XML-based file format for describing encoding conversions, but it appeared to us that it would not be adequate for many situations, notably for legacy encodings that directly encoded presentation forms. In recent months, Martin Hosken and Jonathan Kew have written a specification for a binary file format that can be used to store encoding conversion tables and that will, we believe, provide adequate capability for our encoding conversion needs. Work has been done to develop a set of encoding conversion tools, known as SIL TECkit, based on this binary format. Furthermore, in interactions with members of the Unicode Technical Committee, we were able to present convincing arguments for the need to go beyond their initial designs with the result that the binary mapping table format that Martin and Jonathan designed will likely be compatible with an industry standard for description of encoding conversions developed by Unicode.

3.5 Input

There haven't been major changes here in the past three years. Keyman 3.2 is probably what most SIL users rely upon, though Marc Durdin has released version 4.1, and version 5 is in beta. (It has been in beta for a few months, but Marc has not received enough feedback to feel that he can confidently take it beyond beta.) *Oct. 2000 update: the Keyman 5 beta has advanced considerably in recent months and is now getting quite close to being ready for release.*

The primary change in functionality introduced with version 4 was the ability to run on Windows NT 4, in addition to Windows 9x. (It does not run on Windows 3.x.) Version 5 adds support for Unicode, but runs only on NT (for reasons discussed below). About a year ago, we provided Marc with a feature list we'd like to see. Marc seemed open to considering this, but I don't know what has been done. (I haven't had a chance to install NT to try out version 5.) *Oct. 2000 update: the feature set for version 5 has changed considerably. It is now targeted for Windows 9x/Me, Windows NT 4, and Windows 2000. It supports the use of both 8-bit and Unicode keyboards, though there are some limitations in where Unicode support is available (discussed below).*

Another significant change in version 4 is that Marc followed more closely the facilities provided as part of NLS in Windows 95. It's not clear that this necessarily results in increased functionality for the user, and it may present certain problems. In particular, one implication of this is that every keyboard must be tagged to indicate a particular language. This must be done in terms of LANGIDs (discussed above in §2.2), of which only a certain set have been defined by Microsoft. For most of our language projects, no LANGID would exist. It's possible for someone to define a custom LANGID, but there is no control on this, and there is potential for conflicts. For example, one SIL user recently indicated some inconsistencies arising among teams from SIL and other organisations working in Eurasia, and he easily envisioned a possible scenario in which a user might try to install two keyboards on a system that conflicted with each other due to conflicting LANGIDs. It seems to me that the implications of this aspect of Keyman 4 are not well thought through, and I consider it an area of concern. (Not to fault Marc; I think he did what probably would seem most obvious to a typical developer.) The same issues apply to version 5. *Oct. 2000 update: this issue has been resolved in Keyman 5. A Keyman keyboard can be written to use a specific LANGID when an appropriate one exists for the given language, but if not, then this can be omitted. Keyman will still manage multiple keyboards using Windows' NLS features, but will use a special LANGID that our contact at Microsoft agreed to define to represent languages that are unknown (to Windows).*

A very serious concern has to do with the advent of Unicode. Because of limitations in Unicode support on Windows 9x, those platforms only permit input of Unicode encoded text via 8-bit-to-Unicode mappings through one of the codepages installed on a system, and this must be handled by the application. In many cases, the characters needed by our users are not covered in any of the existing codepages, and it is impossible to define new codepages.

It is possible to redefine existing codepages giving them custom (i.e. hacked) character set definitions. This can be used in conjunction with Keyman 4 to provide input on Windows 9x into applications that support Unicode for some characters that are not supported by standard Windows codepages. Bob Hallissy has succeeded in using this, for example, to provide access to certain uncommon Arabic characters (these characters are in Unicode, but are not supported by any standard codepages). This represents another approach to hacking a solution for particular non-Roman needs in addition to the hacked fonts that we have used for the past ten years, but this is at a rather higher level of complexity (i.e. accessible to fewer), and also has some potentially significant problems associated with it. It may be a viable solution in cases where users needs are nearly but not exactly met by an existing standard codepage, and where users understand and are willing to live with the potential problems. It does not provide any solution in many other cases where standard codepages do not begin to meet users needs, or where large character sets are involved. One example on both accounts is Ethiopic.

The implication of all this is that there is currently no reliable way to generate Unicode-encoded text on Windows 9x with the characters needed by many of our projects. It may be possible to build special workarounds to work with our own software, though this would require special customisations both of the application and of Keyman. It is rather more unlikely that any solution would be available for commercial apps. (Even so, we may come to the conclusion that the costs involved in attempting this might be worthwhile.) For many users, the only alternatives are to stick with 8-bit text or to move to Windows NT (Windows 2000).

Oct. 2000 update: In the spring of this year, we proposed to our contacts at Microsoft a solution to the problem of input on Windows 9x being limited to existing codepages. This solution would not require any change to the OS, and so could work on existing Windows 95 and Windows 98 systems, though it would require special support in application software. Our contacts at Microsoft agreed to this proposal since it addressed a related problem for certain programs when running on Windows NT/2000, and it has since been officially approved. This solution uses a new system message, WM_UNICHAR. This is already supported in some products that are currently in development: Keyman 5, FieldWorks, and in some Microsoft application software. Keyman 5 has been tested with both FieldWorks (WorldPad) and Microsoft software (WordPad) running on Windows 98, and this solution is working. Given the right combinations of software, it is now possible to input any Unicode characters on Windows 9x without requiring codepage support.

Microsoft has made available their “Global IMEs”—input method editors for Chinese, Japanese and Korean that are able to work on any version of Windows. Apparently, it is possible for third parties to build these, but they must be built using standard software development tools, such as C. We don’t know much more than that. Dan Edwards has done some work in this regard. I believe Marc Durdin has wondered whether Keyman should be extended to provide IME functionality, but I don’t think he has done more than raise the question.

On the Mac, Text Services Manager 1.5 (introduced in OS 8.5) provides greater power for creating input methods than was previously available. TSM permits creation of complex input methods that generate 8-bit or Unicode text (though Unicode capability requires specially-designed apps), and is more powerful that Keyman or SILKey. Apple has not yet provided tools for building input methods, however.

SILKey is still available, though with no new functionality. In particular, it is still 8-bit only. *Oct. 2000 update: thus, SILKey is no longer file compatible with the current version of Keyman. At this point, we have no solution for easy creation of Unicode input methods for use on the Mac, and no plans have yet been made to deal with this.*

3.6 Text Analysis

We are a little more aware than we were three years ago of some of the issues that need to be considered for operations such as sorting, searching, case mapping and concatenation, though it still takes up a minimal amount of our attention. (In contrast, work on Graphite has required us to learn much more about line breaking.) This is probably because we still consider the issues of input, encoding and rendering more critical, and because we assume that our SIL application developers have a good handle on these matters. While there may not be a lot that we in NRSI can do directly to provide better support for NR scripts in relation to these operations, we may need to monitor the capabilities that our developers do provide in our apps to ensure they are adequate for the needs of NR scripts.

3.7 Large Scale Publishing

The same systems are being used today as were used three years ago: Ventura is used by most. TeXgX continues to be used in EEG and now by ABS-CS. Additionally, Victor Gaultney was successful in setting up a TeXgX-based system for use by the Tai Dam project team in NAB, and they have been very satisfied users. In contrast, it has proven somewhat difficult to transfer the CC/Ventura technology used for Ethiopic publishing to new users.

Oct. 2000 update: We learned recently that Corel has been facing financial difficulties, and that no further development work for Ventura is currently planned. Thus, the future of Ventura seems quite dubious at this point.

We are seeing major changes on the horizon in the publishing industry, with XML and related technologies being a significant factor. TeX continues to be seen as a powerful layout tool with a lot of potential, and fairly good support for Unicode is reportedly available in Omega (a TeX variant), though it is unclear what support is provided for rendering complex scripts. (There may also be a question as to how well Omega would handle PUA characters.) Lorna Priest, Dennis Drescher and Jonathan Kew are currently doing research in these areas in search of a solution for NR (as well as Roman) publishing in the future.

Adobe has released a new generation of DTP software: InDesign. This application has highly sophisticated text layout capabilities, and, I gather, also makes use of Unicode and of modern smart font technology using OpenType. Since OpenType does not provide all of the required control of rendering behaviour (which, in Microsoft applications is made up for by Uniscribe), Adobe has had to build any such additional rendering support into the application. InDesign does not use Uniscribe; exactly the same rendering behaviour is provided on both Mac and Windows but putting rendering control within the application.

Oct. 2000 update: Adobe has developed a component, known as "CoolType", for providing the necessary rendering behaviour in their applications. This was used in InDesign, and is being added to other Adobe applications, such as Photoshop and FrameMaker.

As of version 1.0, InDesign permits a document to contain any Unicode character, but no input or rendering support is provided for anything other than Roman script. It is likely that better input support will appear. As for complex rendering, however, while it is possible that considerably more support for non-Roman scripts can be added, it is unlikely that Adobe will add support for a wide variety of scripts let alone the kind of flexible rendering support that we really need. (Rendering of non-complex scripts such as Ethiopic should not be a problem, however, although InDesign may not be able to correctly handle some layout features such as optical kerning very well for such a script if it is not an explicitly supported feature.) On the other hand, it might be possible that some of these shortcomings in InDesign can be overcome using custom plug-ins, although the probability of success might be quite low since rendering is likely one aspect of the program that is buried deep in its core.

As of version 1.0, InDesign does not appear to provide any support for XML and related technologies.

Oct. 2000 update: As we have had the opportunity to learn more about InDesign, it has become apparent that, at least in the initial version, this product is not designed for long documents.

Adobe is working on a new version of FrameMaker that will provide support for XML. When it appears, this may represent a good replacement for Ventura, but it will not be released until more than a year from now. Also, Adobe has not shown any particular interest in marketing products for regions using many of the non-Roman scripts that we need to support. At present, the long-term options for large-scale publishing that seems most promising, at least for NR needs, are various TeX-related tools.

Another significant publishing technology from Adobe is the PDF file format and the Acrobat/Distiller software. This has important potential for all types of publishing, from low end through high end, including the web.

3.8 Formalisation of NR-related Requirements for Language Software

As of last spring, we now have a formalised statement—the *LSB Script Strategy* document—of what is required of language software in order to provide adequate support for NR scripts. While this statement is far from perfect, it does represent a valuable gathering of our accumulated knowledge on script-related issues, and provides a useful benchmark and reference for further work.

3.9 Localisation

Whereas we had not given any thought to localisation three years ago, today we have at least realised that it is a potential area of need for SIL software. It was a key component that we included in the *LSB Script Strategy* (of which we were the primary authors), and all of the LSB members concurred that it was an important issue. (Some initially had a negative reaction, but within a few minutes of lively discussion, they were debating not whether it needed to be done but for how many languages it needed to be done.) A key development that raises our awareness of the need is Vision 2025: the implied need to train and equip non-western language workers suggests that we will want to provide software in a variety of languages other than English. (Not hundreds of minority languages, but possibly dozens of languages used for broader communication within particular regions of the world; e.g. French, Spanish, Portuguese, Arabic, Swahili, Amharic, Hausa, Farsi, Hindi, Nepali, Thai, etc.)

We also have a little more awareness of the need for software development practices known in the industry as I18N, "internationalisation". This refers to practices for software design and implementation that make software amenable to being adapted for use in different international markets. Microsoft's ability to adopt a single-binary policy is a benefit they have reaped by adopting good I18N practices. We don't have a lot of experience in this area, though, and may yet need to do a lot of learning.

Since NRSI is not primarily involved in software development, it doesn't particularly fall to us to make the goal of localisable SIL language software be realised. Because of our unique involvement with multilingual- and script-related issues, however, and because script-related capability is an important prerequisite in localisability of software, it was natural for us to include this issue in the LSB Script Strategy. It's remains to be determined in what ways and to what extent NRSI should continue to provide leadership for the organisation in relation to this category of technology.

3.10 Language Identification

Oct 2000 update: As of last February, the issue of language identification was somewhat new for us. It is not strictly a non-Roman script issue, but it affects our work with multilingual data, and also the software systems being developed for use with non-Roman scripts. The issue, in a nutshell, is that information technologies—both software and data standards, such as HTML and XML—use various standards for language identification, and the existing standards do not begin to come close to providing what we need.

I was first confronted by the issue early in 1999 when Keyman 4.0 first appeared, since it required the use of MS Win32 LANGIDs. Later that year, I was contacted by people outside of SIL regarding the possibility of using Ethnologue codes to extend existing standards. By February, I was still learning about the issues. When I discovered that XML uses language identifiers and requires the use of a particular standard, I recognised that this was going to present a problem for us given the importance we expect XML to have for us in the future.

Gary Simons and I co-authored a paper discussing many of the issues, which was presented at the last International Unicode Conference. I have had some discussions with people on the Unicode Technical Committee and also people involved with the International Organization for Standardization (ISO) and the Internet Engineering Task Force regarding possibilities for extending the key standards for language identifiers. There are many who are interested in seeing some major changes, but there are also some who are cautious about change, and some with differing ideas on how changes should be made. There is still considerable work that needs to be done in this area.

The specific problem of using Win32 LANGIDs in Keyman has been resolved in Keyman 5, but the problem of language identifiers in general still remains. For example, in order for a FieldWorks application to instruct Keyman to activate an appropriate keyboard according to the current data field, there needs to be a way to identify what the language of the data is. Again, there is further work that needs to be done in this area.

4. Conclusion

There have been a *lot* of changes in technologies that affect us over the past three years. Some of them, like Unicode, we expected to happen at some point, though we didn't know when. But most of them were things we didn't and probably couldn't anticipate. It will be interesting to see after another three years how much more comes about that we didn't foresee.

What is particularly interesting is the convergence of technologies that make it possible to work with multilingual data and non-Roman scripts using commercial software. The growth of global markets and of the Internet appear to be driving this. The industry is still in transition, so not all of the pieces have fallen into place in all commercial software, but progress is clearly being made.

There are still some areas in which the commercial industry has issues to work through, however. For example, there are several font and rendering issues that have yet to be resolved, such as how to overcome the 64K glyph limit on TrueType fonts (needed for Chinese), how to determine which fonts on a system provide coverage for a given range of Unicode characters (existing mechanisms aren't completely reliable), and especially what should be done about the fact that there are no cross-platform standards for handling complex-script rendering.

The industry is also still focused on relatively major languages. As markets become more global, commercial software is growing to support a wider variety of languages from more countries, but this will not likely reach down to the level of minority languages for a very long time. Furthermore, commercial software is necessarily built on industry standards, and many of our projects involve writing systems that are still nascent and not yet standardised. There have been indications that people in industry may be more open to providing mechanisms for extensibility of behaviours in software, but this can only be considered in ways that are well controlled to ensure the stability of existing implementations and data. This makes commercial develops extremely cautious, and slow to respond. The result is that we will likely need to continue building our own solutions to many needs for some time.

The advances in technology and in commercial software do make things much easier for us, though. For example, there are more cases in which the script-related support that is needed for a language is already provided, and it is vastly easier to mix languages within a given document. Also, in entities where members come from different countries, it is now possible to standardise on one version of Windows and one version of business apps like MS Office that will still allow members to work in the first languages in addition to English and, in many cases, the major or official language of the country in which they are working.

So, there have been exciting developments in the past three years, but we still have a lot of work to be done. Personally, I think this stuff is as interesting and as much fun as it was three years ago. Maybe more so.