Managing change in a digital library system with many interface languages

David Bainbridge, Katrina D. Edgar, John R. McPherson and Ian H. Witten

Department of Computer Science, University of Waikato,
Hamilton, New Zealand.
davidb@cs.waikato.ac.nz
Phone (+64) 7 838 4407, fax (+64) 7 858 5095

Abstract. Managing the organizational and software complexity of a comprehensive open source digital library system presents a significant challenge. The challenge becomes even more imposing when the interface is available in different languages, for enhancements to the software and changes to the interface must be faithfully reflected in each language version. This paper describes the solution adopted by Greenstone, a multilingual digital library system distributed by UNESCO in a trilingual European version (English, French, Spanish), complete with all documentation, and whose interface is available in many further languages. Greenstone incorporates a language translation facility which allows authorized people to update the interface in specified languages. A standard version control system is used to manage software change, and from this the system automatically determines which language fragments need updating and presents them to the human translator.

Keywords: Digital library interfaces, multilingual systems, interface architecture, version control system.

1 INTRODUCTION

As the capabilities of distributed digital libraries increase, managing organizational and software complexity becomes a key issue. How can collections and indexes be updated without impacting queries currently in progress? When users are offered a choice of languages to communicate with the system, how does the software cope with all the versions of the text that is displayed? With multimedia collections, how can search engines for searching various media types be accommodated within a uniform, extensible software framework? How can several user interface clients for the same collection co-exist? How do these clients learn about new collections that are created at other library sites?

This paper focuses on the problem of multilingual interfaces. The Greenstone digital library software [5] is a comprehensive system that is widely used internationally [4], and is freely available under the GNU General Public License. Dozens of collections are hosted at the New Zealand Digital Library site[1] where

[1] http://nzdl.org

the software originates, in a variety of languages and spanning a variety of media formats: text, images, video, audio and music. Many international sites offer further collections. Indeed, the Greenstone mailing list has active members from over forty different countries.

Greenstone is distributed by UNESCO in a trilingual version (English, French, Spanish), complete with all documentation (several hundred pages) in each language. These three language interfaces include all installation instructions, readme files, installer dialogue, on-line help, user interface components, and documentation. The interface is also available in a large number of other languages: Arabic, Chinese, Dutch, German, Hebrew, Indonesian, Italian, Māori, Portuguese, Russian, and Turkish. Other languages, such as Hindi, Kannada, Tamil, Czech, Vietnamese, and Nepalese, are in various stages of development. In addition, Greenstone is a growing system: new facilities and subsystems are constantly being developed and added. As a result, language interfaces run the risk of rapidly become seriously outdated.

The problem of maintaining an evolving multilingual digital library software system is severe—particularly when the software is open source. No single person knows all interface languages; no single person knows about all modifications to the software—indeed there is likely no overlap at all between those who translate the interface and those who develop the software. Currently, Greenstone has about twenty interface languages and there are around 600 linguistic fragments in each interface, ranging from single words like *search*, through short phrases like *search for*, *which contain*, *of the words*, to sentences like *More than ... documents matched the query*, to complete paragraphs like those in the on-line help text. Maintaining the interface in many different languages is a logistic nightmare.

This paper describes the software structure that we have developed to cope with this challenge. We begin by showing some examples of the translation facility to demonstrate its operation. Then we describe the technical infrastructure that provides the underlying functionality. This requires an understanding of the Greenstone macro language—the backbone to the multilingual delivery mechanism—and consequently we give an overview to this also. The translation facility is not specific to macro files, however; it can be adapted to any language management technique—such as Java resource bundles—that records for each language and each item of text to be displayed a pair comprising a language independent label and a language dependent value.

2 TRANSLATOR DEMONSTRATION

The Greenstone translation facility helps users to perform three kinds of task:

- translate the interface into a new language,
- update an existing language interface to reflect new Greenstone facilities,
- refine an existing language interface by correcting errors.

An initial screen, part of which is shown in Figure 1, explains this to users on entry, accompanied by some further information. At the bottom of the page, the

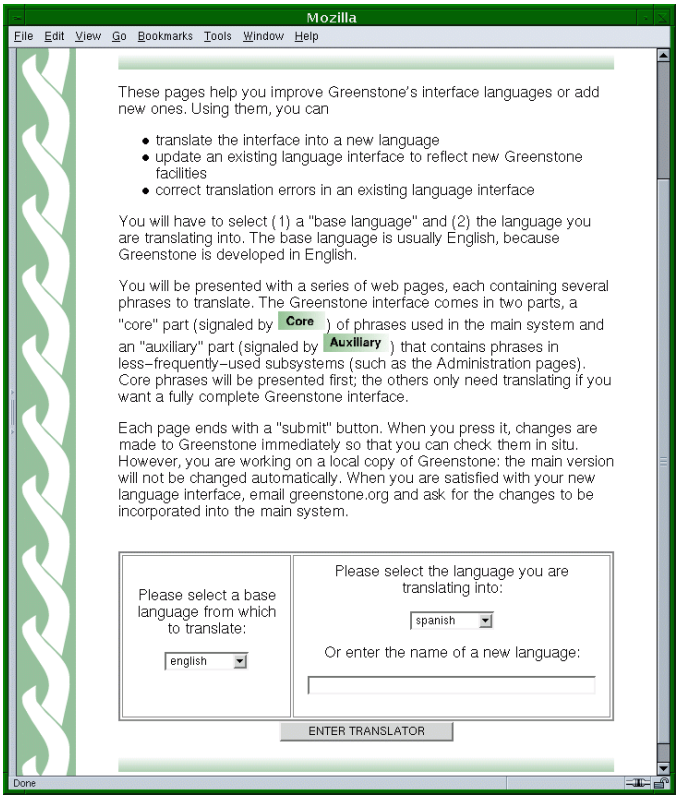


Fig. 1. Part of the translation facility's opening page.

user selects (a) a "base language" and (b) the language they are translating into. The base language is usually English, because this is what is used to develop Greenstone and so it is the most up-to-date interface. However, users are free to select other base languages if they prefer.

Having selected the two languages, users press the "enter translator" button. After a short pause, a page containing phrases to translate appears—generally the first of many such pages. Roughly fifteen phrases appear per page, but the number is contextually reduced for long textual items (which can run to several paragraphs in the case of help text).

2.1 Updating an existing language interface

In Figure 2 the user has begun to update the Spanish language interface, using English as the base language. On the left are the base language phrases, and on the right are boxes into which translated versions can be entered. Two kinds of phrases appear: ones that are missing from the Spanish version, and ones whose

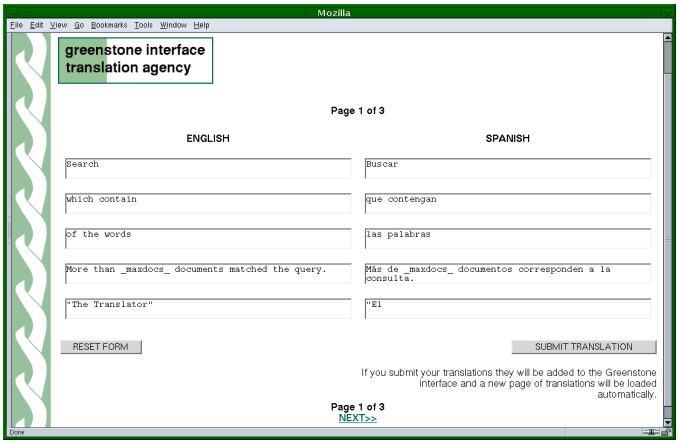


Fig. 2. Translating English phrases to Spanish.

Spanish translation is outdated because the English version has been edited more recently. In the latter case the outdated translation appears as a visual cue. In the figure the user has methodically worked through the phrases and is in the process of entering text into the final box. This entry is particularly noteworthy since it is not a text phrase from the Greenstone interface but instead appears on a button. The translated version will be automatically rendered, on a colored background, by an image manipulation program as described below (Section 3).

When satisfied with all the translations, the user presses the "submit" button shown at the bottom of the page; then the next page in the sequence is generated. Pressing "submit" also commits the changes to the Greenstone system. Changes to the interface take place immediately: users can see their new translations in context by accessing (or reloading) appropriate pages in Greenstone. However, these changes are not made automatically to the public Greenstone site, nor are they automatically committed to the master software repository. Instead, to guard against error and misuse, they take effect in a special replica of the Greenstone site used for translation. If Greenstone encounters any phrases that have not been translated, the fall-back strategy is to render them in the default language for the site, usually English.

If desired, users can reset the page to its initial settings, or proceed to the next page without committing any changes. When satisfied with the entire translation, users notify the administrator of the central Greenstone repository through email of the change. There, issuing a single command fully integrates the changes into the officially released version of the software. If it suits, since each page is saved when it is submitted, a user need not translate all phrases in one sitting. Moreover, when they return to the service the sequence of pages is regenerated, which means that only the outstanding phrases are shown.

Greenstone distinguishes between phrases that are used in the main system—for instance, search, browsing and help pages—and phrases in less-frequently-used subsystems—for instance the Collector used to help build new collections [4], the site administration pages through which usage statistics and logs are viewed, and the translator service itself—for this too needs translating! For well-maintained language interfaces such as Spanish and French, only one or two pages of new translation requests are generated when new features are added. However, some less-used language interfaces contain translations only for the core phrases that appear in the main system.

2.2 Adding new languages

New languages are added in the same way that existing ones are updated, except of course that no existing translations appear in the right-hand column. A total of 68 pages of translations are generated, averaging 9 phrases each (600 in total). About half of these (35 pages, 360 phrases) pertain to the core Greenstone system.

2.3 Text handling

Because of the multilingual nature of Greenstone, attention must be given to character encoding issues. There are many different character encoding schemes in use today—as an example, the code 253 means "capital Y with an acute accent" (Ý) in the standard Western character set (International Organization for Standardization (ISO) standard 8859-1) while the same code corresponds to "a dot-less lower-case i" (ı) in the standard Turkish character set.

All text in Greenstone, including the macro files, is handled internally using Unicode (UTF-8) [3]. Unicode is an ISO standard providing every character in every language with a unique number. For example, in Unicode, a Western Y with acute accent has the code 253 while a Western dot-less i has the code 305. Greenstone supports any character set that can be mapped onto Unicode, which includes the majority of sets currently in use world-wide.

Modern browsers allow Unicode text to be entered into web forms. Unfortunately there is no standard way of forcing a browser to upload information in Unicode—or even to check what character set it uses to submit text fields. Some browsers always submit forms encoded in the user's default character set. However, major browsers generally submit forms using the same character set that is used for the current page, so in practice if pages are sent out with Unicode specified, returned text is usually encoded the same way.

2.4 Refining a language interface

Sometimes phrases in an existing language interface need to be refined. For example, a typographical error may have been overlooked when entering a phrase, or seeing a phrase in its actual interface context may suggest a better form of expression.

To accommodate this requirement users need to be able to locate an existing phrase and update its translation. One solution is to provide a complete list of all phrases in the language, not just the empty or outdated ones presented in Figure 2. The user could scan through this list to locate the desired phrase and correct or revise it. However, given the number of pages and phrases involved this would be a tedious and impractical task.

A more effective strategy is to allow users to locate a given phrase by searching for it, and then receive from the system a page that translates that one phrase. Interestingly, this idea can be implemented by making the set of phrases into a multilingual digital library collection. Within Greenstone, this special collection is designed as follows. Treat each language as a document and each phrase as a section within it. For each document, store its language as metadata and use this as a discriminator to form subcollections. Finally, set the collection to be a "private" (rather than a "public") one to prevent it from appearing on the site's home page. It can still be accessed by a URL that includes the collection name explicitly.

In Figure 3(a) the user is seeking macros in the Spanish language interface that contain the term *autor*. There are five matching items. The user is interested in the fifth, and clicks the page icon to the left of this entry. The result, in Figure 3(b), shows a pseudo-rendered version of the text. Basic HTML formatting of the text (such as the heading) is carried out, but other text phrases embedded in this text phrase are not expanded and appear in their raw form flanked by underscores (_) to indicate that they should not be translated (for example *_textreadingdocs_* in Figure 3(b)). Figure 3(c) shows the page for translating this same phrase. It is brought up by clicking on the "Translate" link in Figure 3(a). The new translation is entered into the right-hand box, or the existing one altered by editing the contents of that box—just as in Figure 2.

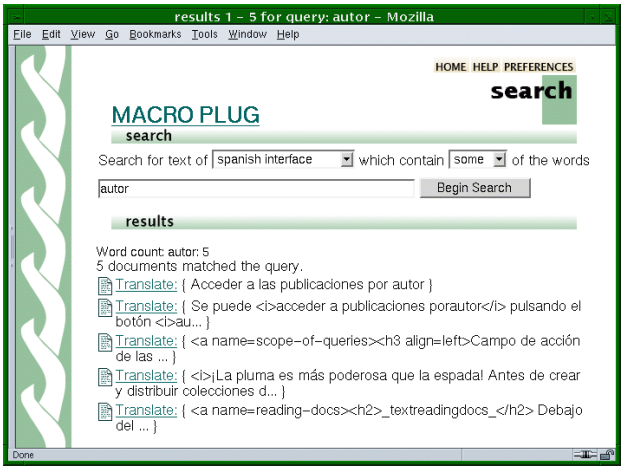
3 GREENSTONE MACROS

Language interface independence in Greenstone is achieved using a simple macro language. Figure 4 shows an artificially-constructed excerpt to illustrate the syntax through which macros are defined and used. Macro definitions comprise a name, flanked by underscores, and the corresponding content, placed within braces ({ ... }).

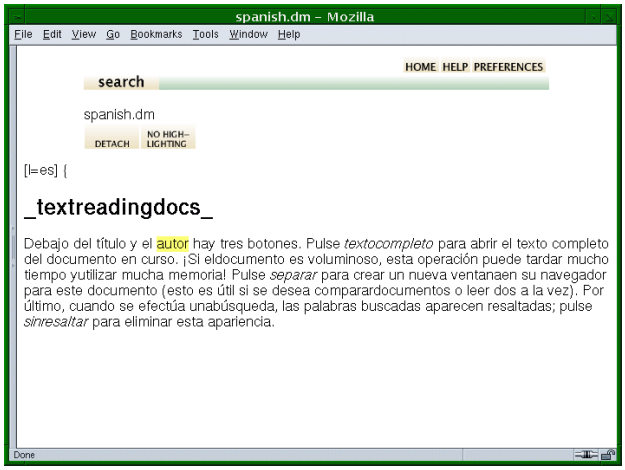
Macros are grouped together into *packages*, with lexical scoping, and an inheritance scheme is used to determine which definitions are in effect at any given time. This allows global formatting styles to be embedded with the particular content that is generated for a page. For example, typical pages contain a *_header_ ... _content_ ... _footer_* sequence. Figure 4 shows a baseline page defined in the "Globals" package, which, in fact, is never intended to be seen. It is overridden in the "query" package below to generate a page that invites the user to enter search terms and perform a query.

Macros can include parameters interposed in square brackets between name and content. Such parameters are known as "page parameters" because they

(a)



(b)



(c)

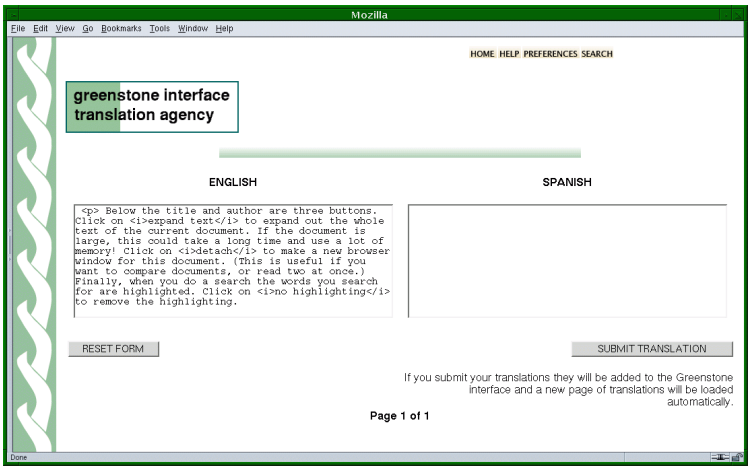


Fig. 3. Various screen shots of the translation interface: (a) seeking Spanish phrases that use the term *autor*; (b) pseudo-rendered version of the fifth text phrase in (a); (c) translation page for the fifth text phrase in (a).

```
package Globals

_header_  { The New Zealand Digital Library Project }
_content_ { Oops. If you are reading this then an error
            has occurred in the runtime system. }
_footer_  { Powered by <a href="www.greenstone.org">Greenstone</a>. }

package query

_content_ { _If_(_cgiargqb_ eq "large",_largequerybox_,_normalquerybox_)
   ... }

# ... the macro descriptions for _largequerybox_, _normalquerybox_,
#  and other nested macros are omitted for brevity

_header_ [l=en] {Begin search }
_header_ [l=fr] {Démarrer la recherche }
_header_ [l=es] {Iniciar la búsqueda}

# ... and so on

# Images containing language-dependent text

## "HELP" ## top_nav_button ## chelp ##
_httpiconchelp_ [l=en,v=1] {_httpimg_/en/chelp.gif}
_httpiconchelp_ [l=en,v=0] {HELP}

# ... and so on
```

Fig. 4. Excerpt of macro file syntax to demonstrate main features.

control the overall generation of a page. They are expressed as $[x = y]$, which gives parameter x the value y. Two parameters of particular interest are l, which determines what language is used, and v, which controls whether or not images are used in the interface.

In Figure 4 three versions of the macro *_header_* are defined within the "query" package, corresponding to the languages English, French and Spanish. They set the language page parameter l to the appropriate two-letter international standard abbreviation (ISO 639), enabling the system to present the appropriate version when the page is generated. If a macro has no definition for a given language, it will resolve to the version given without any language parameter—which, in the current Greenstone implementation, is English (though it needn't be).

Greenstone uses many images that contain language-dependent text. These are created by the GNU image manipulation program GIMP, using scripting to automate the image generation process. Macro files use a specially constructed form of comment (signified by a double hash, ##) to convey additional infor-

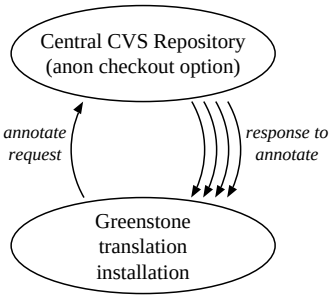


Fig. 5. Relationship between the central repository and translation installation.

mation for a progressing program—in this case an image generation script. An example near the end of Figure 4 generates an icon containing the text *HELP* and places it in the file *chelp.gif* in subdirectory *en* (for "English"). The image generation script parses the comment to determine what type of image to generate, what text is to be placed on it, and where to put the result; automatically generates the image; and stores it in the language-specific directory appropriate to the *l* page parameter (if any).

A precedence ordering for evaluating page parameters is built into the macro language to resolve conflicting definitions. Also included are conditional statements—an example can be seen in the *_content_* macro of Figure 4, which uses an "If" statement, conditioned by the macro *_cgiargqb_*, to determine whether the query box that appears on the search page should be the normal one or a large one. The value of *_cgiargqb_* is set at runtime by the Greenstone system (the user can change it on a "Preferences" page). Many other system-defined macros have values that are determined at runtime: examples include the URL prefix where Greenstone is installed on the system, and the number of documents returned by a search.

Figure 4 is artificial in both content (to demonstration the salient features of the macro language) and layout (for expository convenience). In reality, all English text phrases are stored together in the file *english.dm*, with French phrases in *french.dm* and so on (the suffix *.dm* denotes a macro file). Package names serve to lexically scope the phrase definitions. And in fact there are two files for each language, for example *english.dm* and *english2.dm*, which contain the core Greenstone phrases and those used in auxiliary subsystems respectively, so that the translation facility can differentiate between these two classes.

4 SOFTWARE INFRASTRUCTURE

To help manage software developed by a disparate network of people and promote an open software philosophy, Greenstone utilizes the concurrent versioning system CVS [1]. Version control is used not just for source code, but also for

images, configuration files, and—importantly for this application—the macro files themselves. Macro files define the user interface in a language-independent manner, and also contain the language fragments that flesh out the interface for each particular language. A key insight was that this same mechanism could be used to manage the maintenance of multilingual text. A few customized scripts generate form-based web pages through which the different language interfaces are updated.

Other tools such as *gettext* [2] have been designed to help with localization of text in software projects. However, using the existing macro framework has significant advantages. Changes take effect in the interface immediately; text files are used rather than binary ones; macros can be spread across arbitrary files, and one macro can reference others.

4.1 Generating the translation pages

Figure 5 illustrates a Greenstone installation in which someone is translating or updating a language interface. Requests are sent to the central repository of the Greenstone software, using CVS's *anonymous checkout* option. These requests retrieve the definitive version of each phrase. The *annotate* feature of CVS details for each line in a file when it was last edited, and by whom (amongst other things). This makes it easy for the translation utility to determine which language fragments have been outdated by subsequent alterations to the base language fragments.

Before beginning translation, the user selects the base and target languages using the page in Figure 1. When this page is submitted, a script is activated that performs the steps shown in Figure 6. First, the definitive versions of all text fragments in the two chosen languages are retrieved. Each stream is parsed and built into an associative array, indexed by macro name, that records the date of the most recent edit. These two arrays, one for each language, are analyzed to determine which macro entries in the translation language have never been translated before or whose translation is now out of date. The differences are transferred to a third associative array, also indexed by macro name, which records the content of the base language macro and the target language macro (if present).

From this information a series of pages is generated for the translator. Certain features of the text fragments (for example, length in characters) are used to decide how many phrases to place on each page. The name of the file containing the macro is used to determine whether it is one of the core set of phrases, or forms part of an auxiliary subsystem.

4.2 Updating the repository

One of the central features of CVS is the ability to update one's local copy of the software; that is, to merge any changes made to the central repository into the local version, while maintaining any local changes that are not in the main version. Conversely, it would be easy to fully automate the system by arranging

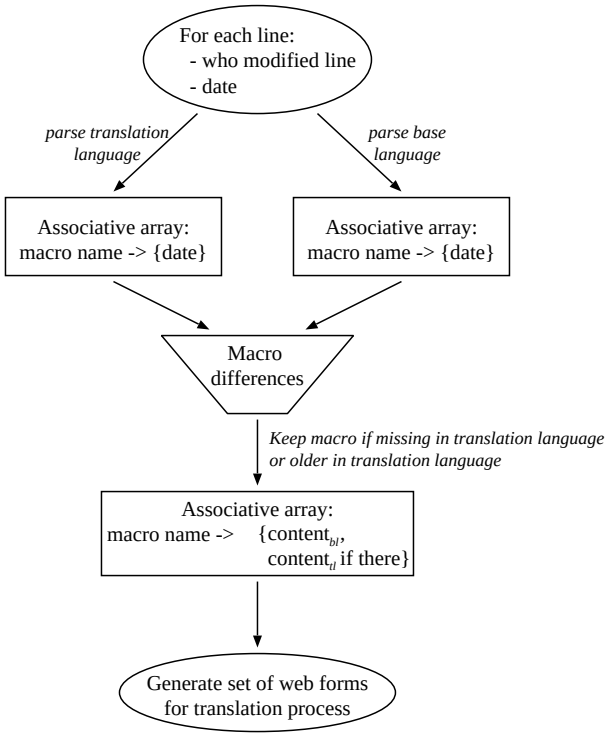


Fig. 6. Determining which macros to present for translation.

that changes in the language macros are automatically committed into the main CVS repository (by trusted users). However, we have chosen not to do this—at least until we gain more experience with the system. Instead, the updated macro files are manually installed by the administrator of the central Greenstone site, who inspects them before committing them to the repository.

4.3 Searching for text phrases

As discussed in Section 2.4 (and illustrated in Figure 3), the Greenstone system itself is used to enable the language phrases to be searched. This is supported by creating a standard digital library collection using the macro files as the input documents. Greenstone uses an extensible scheme of "plugins" to import documents into the system, and a special plugin was written to process macro files. In Greenstone, a configuration file controls which language interfaces are to be offered at any particular site, and this file is parsed to locate the macro files that contain language-specific strings. Each such string is given *Language* metadata based on the filename, using the standard ISO 639 language code.

Appropriate translation pages are generated directly by the search mechanism. The process for generating these is simple. Since the user has signaled that they definitely want to translate this phrase, there is no need to check it against the repository to determine whether it is out of date.

5 CONCLUSIONS

We have presented a mechanism that assists in creating and maintaining multiple language interfaces to a digital library system, in an open-source environment in which the system and its interface are constantly evolving. The problem is a difficult one because many different people are involved—language translators and software developers—who are widely distributed geographically. Furthermore, there is little intersection between the translation group and the development group, in terms both of actual people and the skill sets they possess. Moreover, much of the work involved in translating interfaces is essentially volunteer labor, which is notoriously difficult to organize and control.

The key insight is that the existing version control system used to manage software change can also be recruited to assist with language maintenance. Once a human translator has indicated what language is being considered, and the base language from which translations are made, the system can work out what fragments are new and need to be translated, and also which fragments have been altered since their original translation was made. These can then be presented on a web form for the translator to process. Changes to the interface take place immediately so that the result can be reviewed in context. Two interesting self-referential twists are that the digital library mechanism itself can be used to enable translators to search for specific phrases, and that the interface to the translation system itself can be presented for translation using precisely the same mechanism.

The upshot is that translators are insulated from technical details of how the software is organized, and software developers are insulated from linguistic considerations of how to translate or modify natural language phrases in the interfaces they build and maintain.

References

1. P. Cederqvist. *CVS – Concurrent Versions System*, 1.11.3 edition, December 2002.
2. U. Drepper, J. Meyering, F. Pinard, and B. Haible. *GNU gettext tools*. Free Software Foundation, Inc., Boston, MA, USA, 0.11.2 edition, April 2002.
3. Unicode Consortium. *The Unicode Standard, Version 3.0*. Addison Wesley, Reading, MA, USA, 2000.
4. I. H. Witten and D. Bainbridge. *How to Build a Digital Library*. Morgan Kaufmann, San Francisco, CA, USA, 2003.
5. I. H. Witten, R. J. McNab, S. J. Boddie, and D. Bainbridge. Greenstone: a comprehensive open-source digital library software system. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 113–121, San Antonio, Texas, June 2000.