Greenstone: A Comprehensive Open-Source Digital Library Software System

Ian H. Witten,[*] Rodger J. McNab,[†] Stefan J. Boddie,[*] David Bainbridge[*]

[*] Dept of Computer Science
University of Waikato, New Zealand
E-mail: {ihw, sjboddie, davidb}@cs.waikato.ac.nz

[†] Digilib Systems
Hamilton, New Zealand
E-mail: rodger@digilibs.com

ABSTRACT

This paper describes the Greenstone digital library software, a comprehensive, open-source system for the construction and presentation of information collections. Collections built with Greenstone offer effective full-text searching and metadata-based browsing facilities that are attractive and easy to use. Moreover, they are easily maintainable and can be augmented and rebuilt entirely automatically. The system is extensible: software "plugins" accommodate different document and metadata types.

INTRODUCTION

Notwithstanding intense research activity in the digital library field during the second half of the 1990s, comprehensive software systems for creating digital libraries are not widely available. In fact, the usual solution when creating a digital library is also the most obvious—just put it on the Web. But consider how much effort is involved in constructing a Web site for a digital library. To be effective it needs to be visually attractive and ergonomically easy to use, incorporate convenient and powerful searching capabilities, and offer rich and natural browsing facilities. Above all it must be easy to maintain and augment, which presents a significant challenge if any manual organization is involved.

The alternative is to automate these activities through software tools. But the broad scope of digital library requirements makes this a daunting prospect. Ideally the software should incorporate facilities ranging from multilingual information retrieval to distributed computing protocols, from interoperability to search engine technology, from metadata standards to multiformat document parsing, from multimedia to multiple operating systems, from Web browsers to plug-and-play DVDs.

The Greenstone Digital Library Software from the New Zealand Digital Library (NZDL) project tackles this issue by providing a new way of organizing information and making it available over the Internet. A *collection* of information comprises several (typically several thousand, or several million) *documents*, and a uniform interface is provided to all documents in a collection. A library may include many different collections, each organized differently—though there is a strong family resemblance in how collections are presented.

Making information available using this system is far more than "just putting it on the Web." The collection becomes maintainable, searchable, and browsable. Each collection, prior to presentation, undergoes a "building" process that, once established, is completely automatic. This process creates all the structures that are used at run-time for accessing the collection. Searching is based on various indexes, while browsing is based on various metadata; support structures for both are created during the building operation. When new material appears it can be fully incorporated into the collection by rebuilding.

To address the exceptionally broad demands of digital libraries, the system is public and extensible. It is issued under the Gnu public license and, in the spirit of open-source software, users are invited to contribute modifications and enhancements. Only through an international cooperative effort will digital library software become sufficiently comprehensive to meet the world's needs. Currently the Greenstone software is used at sites in Canada, Germany, New Zealand, Romania, UK, and the US, and collections range from newspaper articles to technical documents, from educational journals to oral history, from visual art to folksongs. The software has been used for collections in many different languages, and for CD-ROMs that have been published by the United Nations and other humanitarian agencies in Belgium, France, Japan, and the US for distribution in developing countries (Humanity Libraries, 1998; PAHO, 1999; UNESCO, 1999; UNU, 1998). Further details can be obtained from *www.nzdl.org*.

Figure 1: Searching the HDL collection

This paper sets the scene with a brief discussion of what a digital library is. We then give an overview of the facilities offered by Greenstone and show how end users find information in collections. Next we describe the files and directories involved in a collection, and then discuss the processes of updating existing collections and creating new ones, including extending the software to provide new facilities. We conclude with an overview of related work.

WHAT IS A DIGITAL LIBRARY?

Ten definitions of the term "digital library" have been culled from the literature by Fox (1998), and their spirit is captured in the following brief characterization:

> *A collection of digital objects, including text, video, and audio, along with methods for access and retrieval, and for selection, organization and maintenance of the collection*

(Akscyn and Witten, 1998). Lesk (1998) views digital libraries as "organized collections of digital information," and wisely recommends that they articulate the principles governing what is included and how the collection is organized.

Digital libraries are generally distinguished from the World-Wide Web, the essential difference being in selection and organization. But they are not generally distinguished from a web *site*: indeed, virtually all extant digital libraries manifest themselves as a web site. Hence the obvious question: to make a digital library, why not just put the information on the Web?

But we make a distinction between a digital library and a web site that lies at the heart of our software design: one should easily be able to add new material to a library without having to integrate it manually or edit its content in any way. Once added, new material should immediately become a first-class component of the library. And what permits it to be integrated into existing searching and browsing structures without any manual intervention is *metadata*. This provides sufficient focus to the concept of "digital library" to support the development of a construction kit.

OVERVIEW OF GREENSTONE

Information collections built by Greenstone combine extensive full-text search facilities with browsing indexes based on different metadata types. There are several ways for users to find information, although they differ between collections depending on the metadata available and the collection design. Typically you can *search for particular words* that appear in the text, or within a section of a document, or within a title or section heading. You can *browse documents by title*: just click on the displayed book icon to read it. You can *browse documents by subject*. Subjects are represented by bookshelves: just click on a shelf to see the books. Where appropriate, documents come complete with a table of contents (constructed automatically): you can click on a chapter or subsection to open it, expand the full table of contents, or expand the full document.

An example of searching is shown in Figure 1 where documents in the Global Help Project's Humanity Development Library (HDL) are being searched for chapters matching the word *butterfly*. In Figure 2 the same collection is being browsed by subject: by clicking on the bookshelf icons the user has discovered an item under Section 16, Animal Husbandry. Pursuing an interest in butterfly farming, the user selects a book by clicking on its book icon. In Figure 3 the front cover of the book is displayed as a graphic on the left, and the automatically constructed table of contents appears at the start of the document. The current focus, *Introduction and Summary*, is shown in bold in the table of contents with its text starting further down the page.

In accordance with Lesk's advice, a statement of purpose and coverage accompanies each collection, along with an explanation of how it is organized (Figure 1 shows the start of this). A distinction is made between *searching* and *browsing*. Searching is full-text, and—depending on the collection's design—the user can choose between indexes built from different parts of the documents, or from different metadata. Some collections have an index of full documents, an index of sections, an index of paragraphs, an index of titles, and an index of section headings, each of which can be searched for particular words or phrases. Browsing involves data structures created from metadata that the user can examine: lists of authors, lists of titles, lists of dates, hierarchical classification structures, and so on. Data structures for both browsing and searching are built according to instructions in a configuration file, which controls both building and serving the collection. Sample configuration files are discussed below.

Figure 2: Browsing the HDL collection by subject

Rich browsing facilities can be provided by manually linking parts of documents together and building explicit indexes and tables of contents. However, manually-created linking becomes difficult to maintain, and often falls into disrepair when a collection expands. The Greenstone software takes a different tack: it facilitates *maintainability* by creating all searching and browsing structures automatically from the documents themselves. No links are inserted by hand. This means that when new documents in the same format become available, they can be added automatically. Indeed, for some collections this is done by processes that wake up regularly, scout for new material, and rebuild the indexes—all without manual intervention.

Collections comprise many documents: thousands, tens of thousands, or even millions. Each document may be hierarchically organized into *sections* (subsections, sub-subsections, and so on). Each section comprises one or more *paragraphs*. Metadata such as author, title, date, keywords, and so on, may be associated with documents, or with individual sections of documents. This is the raw material for indexes. It must either be provided explicitly for each document and section (for example, in an accompanying spreadsheet) or be derivable automatically from the source documents. Metadata is converted to Dublin Core and stored with the document for internal use.

In order to accommodate different kinds of source documents, the software is organized so that "plugins" can be written for new document types. Plugins exist for plain text documents, HTML documents, email documents, and bibliographic formats. Word documents are handled by saving them as HTML; PostScript ones by applying a preprocessor (Nevill-Manning *et al*., 1998). Specially written plugins also exist for proprietary formats such as that used by the BBC archives department. A collection may have source documents in different forms: it is just a matter of specifying all the necessary plugins. In order to build browsing indexes from metadata, an analogous scheme of "classifiers" is used: classifiers create indexes of various kinds based on metadata. Source documents are brought into the Greenstone system through a process called *importing*, which uses the plugins and classifiers specified in the collection configuration file.

The international Unicode character set is used throughout, so documents—and interfaces—can be written in any language. Collections have so far been produced in English, French, Spanish, German, Maori, Chinese, and Arabic. The NZDL Web site provides numerous examples. Collections can contain text, pictures, and even audio and video clips; a text-only version of the interface is also provided to accommodate visually impaired users. Compression technology is used to ensure best use of storage (Witten *et al*., 1999). Most non-textual material is either linked to textual documents or accompanied by textual descriptions (such as photo captions) to allow full-text searching and browsing. However, the architecture permits the implementation of plugins and classifiers even for non-textual data.

The system includes an "administrative" function whereby specified users can examine the composition of all collections, protect documents so that they can only be accessed by registered users on presentation of a password, and so on. Logs of user activity are kept that record all queries made to every Greenstone collection (though this facility can be disabled).

Although primarily designed for Internet access over the World-Wide Web, collections can be made available, in precisely the same form, on CD-ROM. In either case they are accessed through any Web browser. Greenstone CD-ROMs operate on a standalone PC under Windows 3.X, 95, 98, and NT, and the interaction is identical to accessing the collection on the Web—except that response is faster and more predictable. The requirement to operate on early Windows systems is one that plagues the software design, but is crucial for many users—particularly those in underdeveloped countries seeking access to humanitarian aid collections. If the PC is connected to a network (intranet or Internet), a custom-built Web server provided on each CD makes exactly the same information available to others through their standard Web browser. The use of compression ensures that the greatest possible volume of information can be packed on to a CD-ROM.

The collection-serving software operates under Unix and Windows NT, and works with standard Web servers. A flexible process structure allows different collections to be served by different computers, yet be presented to the user in the same way, on the same Web page, as part of the same digital library, even as part of the same collection (McNab and Witten, 1998). Existing collections can be updated and new ones brought on-line at any time, without bringing the system down; the process responsible for the user interface will notice (through periodic polling) when new collections appear and add them to the list presented to the user.

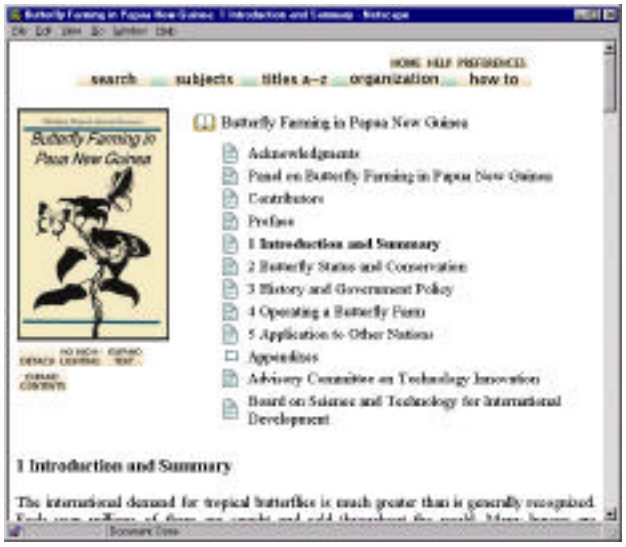

Figure 3: Reading a book in the HDL

FINDING INFORMATION

Greenstone digital library systems generally include several separate collections. A home page allows you to select a collection; in addition, each collection has its own "about" page that gives you information about how the collection is organized and the principles governing what is included.

All icons in the screenshots of Figures 1–4 are clickable. Those icons at the top of the page return to the home page, provide help text, and allow you to set user interface and searching preferences. The navigation bar underneath gives access to the searching and browsing facilities, which differ from one collection to another.

Each of the five buttons provides a different way to find information. You can *search for particular words* that appear in the text from the "search" page (or from the "about" page of Figure 1). This collection contains indexes of chapters, section titles, and entire books. The default search interface is a simple one, suitable for casual users; advanced searching—which allows full Boolean expressions, phrase searching, case and stemming control—can be enabled from the *Preferences* page.

This collection has four browsable metadata indexes. You can *access publications by subject* by clicking the *subjects* button, which brings up a list of subjects, represented by bookshelves (Figure 2). You can *access publications by title* by clicking *titles a-z* (Figure 4), which brings up a list of books in alphabetic order. You can *access publications by organization* (i.e. Dublin Core "publisher"), bringing up a list of organizations. You can *access publications by "how to" listing*, yielding a list of hints defined by the collection's editors. We use the Dublin Core as a base and extend it in an *ad hoc* manner to accommodate the individual requirements of collection designers.

FILES IN A COLLECTION

When a new collection is created or material is added to an existing one, the original source documents are first brought into the system through a process known as "importing." This involves converting documents into a simple HTML-like format known as GML (for "Greenstone Markup Language"), which includes any metadata associated with the document. Documents are assumed to be in the Unicode UTF-8 code (of which the ASCII characters form a subset).

Files and directories

There is a separate directory for each collection, which contains five subdirectories: the original raw material (*import*), the GML files created from this (*archives*), the final collection as it is served to users (*index*), a directory for use during the building process (*building*), and one for any supporting files (*etc*)—including the configuration file that controls the collection creation procedure. Additional files might be required: for example, building a hierarchy of classifications requires a data file of sub-classifications.

The imported documents

In order to identify documents internally, a unique object identifier or OID is assigned to each original source document when it is imported (formed by hashing the content, to overcome file duplication effects caused by mirroring) and stored as metadata within that document. It is important that OIDs persist throughout the index-building process—so that a user's search history is unaffected by rebuilding the collection. OIDs are assigned by hashing the contents of the original source document.

Once imported, each document is stored in its own subdirectory of *archives*, along with any associated files—for example, images. To ensure compatibility with Windows 3.0, only eight characters are used in directory and file names, which causes annoying but essentially trivial complications.

Inside the documents

The GML format imposes a limited amount of structure on documents. Documents are divided into paragraphs. They can be split hierarchically into sections and subsections. OIDs are extended to identify these components by appending numbers, separated by periods, to a document's OID. When a book is read, its section hierarchy is visible as the table of contents (Figure 3). Chapters, sections, subsections, and pages are all implemented simply as "sections" within the document. In some collections documents do not have a hierarchical subsection structure, but are split into pages to permit browsing within a retrieved document.

The document structure is used for searchable indexes. There are three levels of index: *documents*, *sections*, and

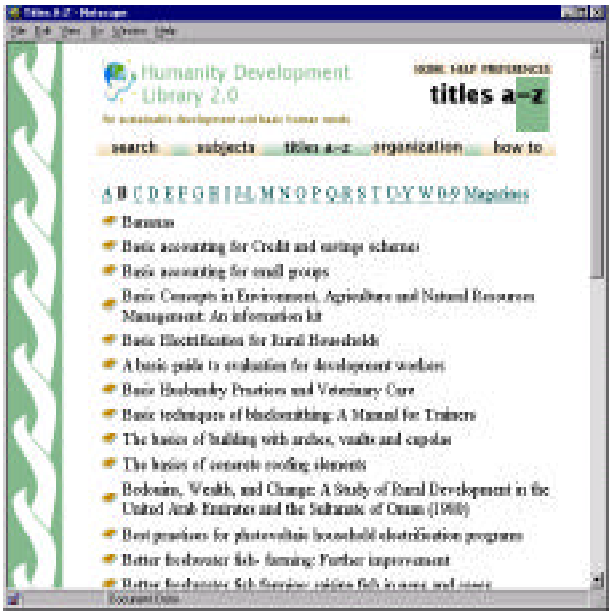

Figure 4: Browsing titles in the HDL

paragraphs, corresponding to the distinctions that GML makes—the hierarchical structure is flattened for the purposes of creating these indexes. Indexes can be of text, or metadata, or any combination. Thus you can create a searchable index of section titles, and/or authors, and/or document descriptions, as well as the document text.

UPDATING EXISTING COLLECTIONS

Updating an existing collection with new files in the same format is easy. For example, the raw material for the HDL is supplied in the form of HTML files marked up with <<TOC>> tags to split books into sections and subsections, and <<I>> tags to indicate where an image is to be inserted. For each book in the library there is a directory that contains a single HTML file representing the book, and separate files containing the associated images. An accompanying spreadsheet file contains the classification hierarchy; this is converted to a simple file format (using Excel's *Save As* command).

Since the collection exists, its directory is already set up with subdirectories *import*, *archives*, *building*, *index*, and *etc*, and the *etc* directory will contain a suitable collection configuration file.

The updating procedure

To update a collection, the new raw material is placed in the *import* directory, in whatever form it is available. Then the *import* process is invoked, which converts the files into GML using the specified plugins. Old material for which GML files have previously been created is not re-imported. Then the *build* process is invoked to build the requisite indexes for the collection. Finally, the contents of the *building* directory are moved into the *index* directory, and the new version of the collection automatically becomes live.

This procedure may seem cumbersome. But all the steps are necessary for efficient operation with large collections. The *import* process could be performed on the fly during the building operation—but because building indexes is a multipass operation, the often lengthy importing would be repeated several times. The *build* process can take considerable time—a day or two, for very large collections. Consequently, the results are placed in the *building* directory so that, if the collection already exists, it will continue to be served to users in its old form throughout the building operation.

Active users of the collection will not be disturbed when the new version becomes live—they will probably not even notice. The persistent OIDs ensure that interactions remain coherent—users who are examining the results of a query or browse operation will still retrieve the expected documents—and if a search is actually in progress when the change takes place the program detects the resulting file-structure inconsistency and automatically and transparently re-executes the query, this time on the new version of the collection.

How it works

The original material in the *import* directory may be in any format, and plugins are required to process each format type. The plugins that a collection uses must be specified in the collection configuration file. The *import* program reads the list of plugins and passes each document to each plugin in order until it finds one that can process it. When updating an existing collection, all plugins necessary to process new material should already have been specified in the configuration file.

The building step creates the indexes for both searching and browsing. The MG software is generally used to do the searching (Witten *et al.*, 1999), and the *mgbuild* module is automatically invoked to create each of the indexes that is required. For example, the Humanity Development Library has three indexes, one for entire books, one for chapters, and one for section titles. Subdirectories of the *index* directory are created for each of these indexes.

(a)

```
creator                 davidb@cs.waikato.ac.nz                          1
maintainer              davidb@cs.waikato.ac.nz                          2
public                  True                                             3
                                                                         4
indexes                 document:text                                    5
defaultindex            document:text                                    6
plugins                 GMLPlug TEXTPlug ArcPlug RecPlug                 7
                                                                         8
classify                AZList metadata=Title                            9
                                                                         10
collectionmeta          collectionname    "generic text collection"      11
collectionmeta          .document:text    "documents"                    12
```

(b)

```
creator                 davidb@cs.waikato.ac.nz                          1
maintainer              davidb@cs.waikato.ac.nz                          2
public                  True                                             3
                                                                         4
indexes                 document:text document:From                      5
defaultindex            document:text                                    6
plugins                 GMLPlug EMAILPlug ArcPlug RecPlug                7
                                                                         8
classify                AZList metadata=Title                            9
classify                DateList                                         10
                                                                         11
collectionmeta          collectionname    "Email messages"               12
collectionmeta          .document:text    "documents"                    13
collectionmeta          .document:From    "email senders"                14
                                                                         15
format                  QueryResults \                                   16
                        <td>[link][icon][/link]</td><td>[Title]</td><td>[Author]</td>   17
```

Figure 5: Collection configuration files (a) generic, (b) for an email collection

MG also compresses the text of the collection; and the image files are linked into the *index* subdirectory. Now none of the material in the *import* and *archives* directories is needed to run the collection and can be removed from the file system (though they would be needed if the collection were rebuilt).

Associated with each collection is a database stored in GDBM (Gnu database manager) format. This contains an entry for each document, giving its OID, its internal MG document number, and metadata such as title. Information for each of the browsing indexes, which appear as buttons on the Greenstone search/browse bar, is also extracted during the building process and stored in the database. A "classifier" program is required for each browsing index to extract the appropriate information from GML documents. Like plugins, classifiers are written on an *ad hoc* basis for the particular information required, and where possible reused from one collection to another.

The building program creates the indexes based on whatever appears in the *archives* directory. The first plugin specified by all collections is one that processes GML files, and so if *archives* contains imported files they will be processed correctly. If it contains material in the original format, that will be converted using the appropriate plugin. Thus the import process is optional.

GML is designed to be fast and easy to parse, an important requirement when millions of documents are to be processed. Something as simple as requiring tags to be lower-case, for example, yields a substantial speed-up. In certain circumstances, however, it might be preferable to use a standardized format such as XML. This is straightforward to implement¾just write an XML plugin¾although we have not done so ourselves. Given the transitory nature of the imported data, to date, we have found GML a satisfactory and beneficial format.

CREATING NEW COLLECTIONS

Building new collections from scratch is only slightly different from updating an existing collection. The key new requirement is creating a collection configuration file, and a software utility is provided to help. Two pieces of information are required for this: the name of the directory that the collection will use (into which the source data and other files will eventually be placed), and a contact e-mail address for use if any problems are encountered by the software once the collection is up and running. The utility creates files and directories within the newly-named directory to support a generic collection of plain text documents. With suitable data placed in the *import* directory, building the collection at this point will yield a document-level searchable index of all the text and a browsable list of "titles" (defined in this case to be the document filenames).

To enhance the functionality and presentation— something anything but the most trivial collection will require—the configuration file must be edited. For a collection sourced from documents in an already supported data format, presented in a similar fashion to an existing collection, the

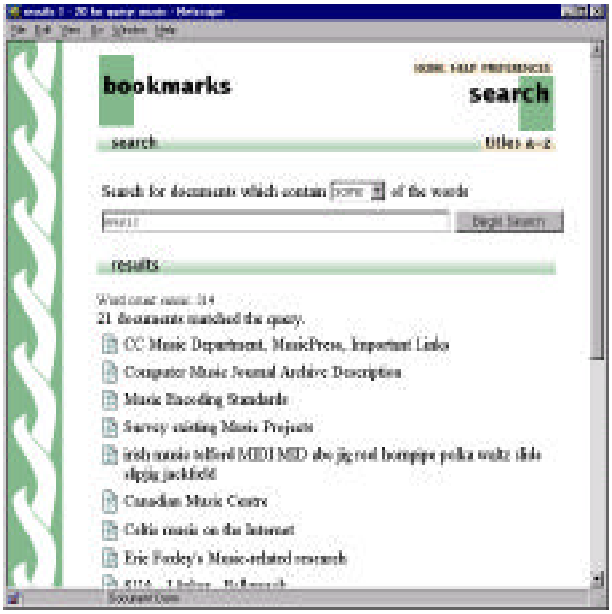

Figure 6: Searching bookmarked Web pages

amount of editing is minimal. Importing new data formats and browsing metadata in ways not currently supported are more complex activities that require programming skills.

Modifying the configuration file

Figure 5b shows simple alterations to the generic configuration file in Figure 5a that was generated by the new-collection utility. *TEXTPlug* is replaced with *EMAILPlug* (line 7) which reads email files and extracts metadata (*From*, *To*, *Date*, *Subject*) from them. A classifier for dates is added (line 10) to make the collection browsable chronologically. The default presentation of search results is overridden (line 17) to display both the title of the message (i.e. Dublin Core *Title*) and its sender (i.e. Dublin Core *Author*). Elements in square brackets, such as *[Title]*, are replaced by the metadata associated with a particular document. The built-in term *[icon]* produces a suitable image that represents the document (such as a book icon or page icon), and the *[link]…[/link]* construct forms a hyperlink to the complete document. Anything else in the format statement, which in this case is solely table-cell tags in HTML, is passed through to the page being displayed.

As this example shows, creating a new collection that stays within the bounds of the library's established capabilities falls within the capability of many computer users—for instance, computer-trained librarians. Extending Greenstone to handle new document formats and browse metadata in new ways is more challenging.

Writing new plugins and classifiers

Extensibility is obtained through plugins and classifiers. These are modules of code that can be slotted into the system to enhance its capabilities. Plugins parse documents, extracting the text and metadata to be indexed. Classifiers control how metadata is brought together to form browsable data structures. Both are specified in an object-oriented framework using inheritance to minimize the amount of code written.

A plugin must specify three things: what file formats it can handle, how they should be parsed, and whether the plugin is recursive. File formats are normally determined using regular expression matching on the filename. For example, the HTML plugin accepts all files that end in *.htm*, *.html*, *.HTM*, or *.HTML*. (It is quite possible, however, to write plugins that "look inside" the file as well.) For other files, the plugin returns *undefined* and the file is passed to the next plugin in the collection's configuration file (e.g. Figure 5 line 7). If it can, the plugin parses the file and returns the number of documents processed. This involves extracting text and metadata and adding it to the library's content through calls to *add text* and *add metadata*.

Some plugins ("recursive" ones) add extra files into the stream of data processed during the building phase by artificially reactivating the list of plugins. This is how directory hierarchies are traversed.

Plugins are small modules of code that are easy to write. We monitored the time it took to develop a new one that was different to any we had produced so far. We chose to make as an example a collection of HTML bookmark files, the motivation being to produce a convenient way of searching and browsing one's bookmarked Web pages. Figure 6 shows a user searching for bookmarked pages about *music*. The new plugin took under an hour to write, and was 160 lines long (ignoring blank lines and comments)—about the average length of existing plugins.

Classifiers are more general than plugins because they work on GML-format data. For example, any plugin that generates date metadata in accordance with the Dublin core can request the collection to be browsable chronologically by specifying the *DateList* classifier in the collection's configuration file (Figure 7). Classifiers are more elaborate than most plugins, but new ones are seldom required. The average length of existing classifiers is 230 lines.

Classifiers must specify three things: an initialization routine, how individual documents are classified, and the final browsable data structure. Initialization takes care of any options specified in the configuration file (such as *metadata=Title* on line 9 of Figure 5b). Classifying individual documents is an iterative process: for each one, a call to *document-classify* is made. On presentation of the document's OID, the necessary metadata is located and used to control where the document is added to the browsable data structure being constructed.

Once all documents have been added, a request is made for the completed data structure. Some classifiers return the data structure directly; others transform the data structure before it is returned. For example, the *AZList* classifier

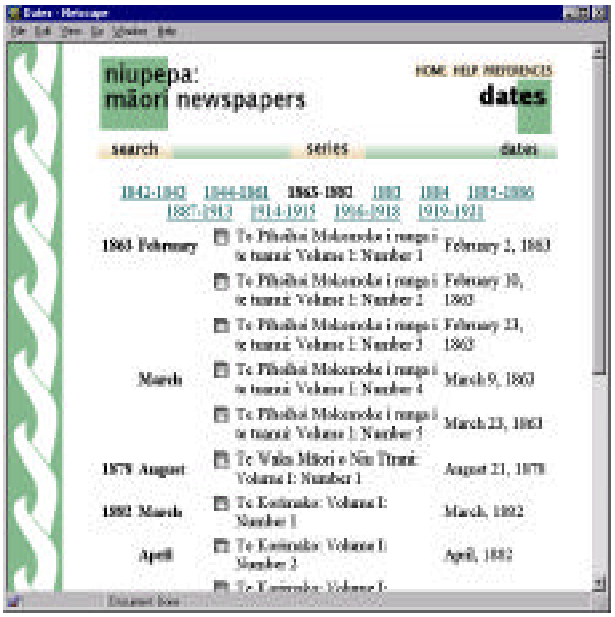

Figure 7: Browsing a newspaper collection by date

divides the alphabetically sorted list of metadata into separate pages of about the same size and returns the alphabetic ranges for each one (Figure 4).

OVERVIEW OF RELATED WORK

Two projects that provide substantial open source digital library software are Dienst (Lagoze and Fielding, 1998) and Harvest (Bowman *et al.*, 1994). The origins of Dienst (*www.cs.cornell.edu/cdlrg*) stretch back to 1992. The term has come to represent three entities: a conceptual architecture for distributed digital libraries; an open protocol for service communication; and a software system that implements the protocol. To date, five sample digital libraries have been built using this technology. They manifest themselves in two forms: technical reports and primary source documents.

Best known is NCSTRL, the Networked Computer Science Technical Reference Library project (*www.ncstrl.org*). This collection facilitates searching by title, author and abstract, and browsing by year and author, across a distributed network of document repositories. Documents can (where supported) be delivered in various formats such as PostScript, a thumbnail overview of the pages, and a GIF image of a particular page.

The *Making of America* resource is an example of a collection based around primary sources¾in this case American social history, 1830−1900. It has a different "look and feel" to NCSTRL, being strongly oriented toward browsing rather than searching. A user navigates their way through a hierarchical structure of hyperlinks to reach a book of interest. The book itself is a series of scanned images: delivery options include going directly to a page number, next and previous page buttons, and displaying a particular page at different resolutions. A text version of the page is also available upon which a searching option is also provided.

Started in 1994, Harvest is also a long-running research project. It provides an efficient means of gathering source data from the Internet and distributing indexing information over the Internet. This is accomplished through five components: *gatherer*, *broker*, *indexer*, *replicator* and *cache*. The first three are central to creating, updating and searching a collection; the last two help to improve performance over the Internet through transparent mirroring and caching techniques.

The system is configurable and customizable. While searching is most commonly implemented using Glimpse (*glimpse.cs.arizona.edu*), in principle any search engine that supports incremental updates and Boolean combinations of attribute-based queries can be used. It is possible to control what type of documents are gathered during creation and updating, and how the query interface looks and is laid out.

Sample collections cited by the developers include 21,000 computer science technical reports and 7,000 home pages. Other examples include a sizable collection of agriculture-related electronic journals and magazines called "tomato-juice" (accessed through *hegel.lib.ncsu.edu*) and a full-text index of library-related electronic serials (*sunsite.berkeley.edu/IndexMorganagus*). Harvest is also often used to index Web sites (for example *www.middlebury.edu*).

Comparing Greenstone with Dienst and Harvest, there are both similarities and differences. All provide substantial digital library systems, hence common themes recur, but they are driven by projects with different aims. Harvest, for instance, was not conceived as a digital library project at all, but by virtue of its selective document gathering process it can be classed (and is used) as one. While it provides sophisticated search options, it lacks the complementary service of browsing. Furthermore it adds no structure or order to the documents collected, relying on whatever structures are present in the site that they were gathered from. A proven strength of the design is its flexibility through configuration and customization¾an element also present in Greenstone.

Dienst¾best exemplified through the NCSTRL work¾supports searching and browsing, like Greenstone. Both use open protocols. Differences include a high reliance in Dienst on user-supplied information when a document is added, and a smaller range of document types supported—although Dienst does include a document model that should, over time, allow this to expand with relative ease.

There are also commercial systems that provide similar digital library services to those described. However, since

corporate culture instills proprietary attitudes there is little opportunity for advancement through a shared collaborative effort. Consequently they are not reviewed here.

CONCLUSIONS

Greenstone is a comprehensive software system for creating digital library collections. It builds data structures for searching and browsing from the material provided, rather than relying on any hand-crafting. The process is controlled by a configuration file, and once a collection exists new material can be added completely automatically. Browsing is based on Dublin Core metadata.

New collections can be developed easily, particularly if they resemble existing ones. Extensibility is achieved through software "plugins" that can be written to accommodate documents, and metadata, in different formats. Standard plugins exist for many document types; new ones are easily written. Browsing is controlled by "classifiers" that process metadata into browsing structures (by date, alphabetical, hierarchical, etc).

However, the most powerful support for extensibility is achieved not by technical means but by making the source code freely available under the Gnu public license. Only through an international cooperative effort will digital library software become sufficiently comprehensive to meet the world's needs with the richness and flexibility that users deserve.

ACKNOWLEDGMENTS

We gratefully acknowledge all those who have worked on the Greenstone software, and all members of the New Zealand Digital Library project for their enthusiasm and ideas.

REFERENCES

1. Akscyn, R.M. and Witten, I.H. (1998) "Report on First Summit on International Cooperation on Digital Libraries." ks.com/idla-wp-oct98.
2. Bowman, C.M., Danzig, P.B., Manber, U., and Schwartz, M.F. "Scalable Internet resource discovery: Research problems and approaches" *Communications of the ACM*, Vol. 37, No. 8, pp. 98−107, 1994.
3. Fox, E. (1998) "Digital library definitions." ei.cs.vt.edu/~fox/dlib/def.html.
4. Humanity Libraries (1998) *Humanity Development Library*. CD-ROM produced by the Global Help Project, Antwerp, Belgium.
5. Lagoze, C. and Fielding, D "Defining Collections in Distributed Digital Libraries" *D-Lib Magazine*, Nov. 1998.
6. PAHO (1999) *Virtual Disaster Library*. CD-ROM produced by the Pan-American Health Organization, Washington DC, USA.
7. McNab, R.J., Witten, I.H. and Boddie, S.J. (1998) "A distributed digital library architecture incorporating different index styles." *Proc IEEE Advances in Digital Libraries*, Santa Barbara, CA, pp. 36–45.
8. Nevill-Manning, C.G., Reed, T., and Witten, I.H. (1998) "Extracting text from PostScript" *Software—Practice and Experience*, Vol. 28, No. 5, pp. 481–491; April.
9. UNESCO (1999) *SAHEL point DOC: Anthologie du développement au Sahel*. CD-ROM produced by UNESCO, Paris, France.
10. UNU (1998) *Collection on critical global issues.* CD-ROM produced by the United Nations University Press, Tokyo, Japan.
11. Witten, I.H., Moffat, A. and Bell, T. (1999) *Managing Gigabytes: compressing and indexing documents and images*, Morgan Kaufmann, second edition.