Dynamic digital library construction and configuration

David Bainbridge, Katherine J. Don, George R. Buchanan, Ian H. Witten
Steve Jones, Matt Jones, and Malcolm I. Barr

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. This paper describes a digital library architecture and implementation that is configurable, extensible and dynamic in the way it presents content and in the services it provides. The design manifests itself as a network of modules that communicate in terms of XML messages. All modules characterize the functionality they implement in response to a "describe yourself" message, and can transform messages using XSLT to support different levels of configurability. Traditional library values such as backwards compatibility and multiplatform operation are combined with the ability to add new collections and services adaptively. The paper describes the new design and shows how it can be used to build four different digital library systems. We conclude by showing how the design fits existing interoperability frameworks.

Keywords: Digital library services, dynamic configurability, XML, XSL Transforms

1 Introduction

This paper describes a digital library design that improves upon the Greenstone toolkit [7]. First, it provides more flexible ways of dynamically configuring the run-time system and adding new services to it. Second, it lowers the overhead incurred by collection developers when accessing this flexibility to organize and present their content. Third, it modularizes the internal structure and simplifies the addition of new modules. The design is based on widely-accepted standards such as XML, current software practices such as simple protocols (like SOAP), cross-platform development strategies (Java), and contemporary schemes for software modularization and dynamic updates. Most important of all, it is informed by our experience with the current Greenstone system and the problems and challenges faced by real users, international collection developers, and practicing librarians.

The structure of the paper is as follows. First we give some background out of which the requirements for the digital library software arose. Next we describe the new design, called Greenstone3, and discuss how it meets the identified needs. Fundamental to the approach is the use of XML throughout for data representation, combined with XSL Transforms to provide a flexible mechanism for adjusting the functionality of the runtime system without having to modify and recompile the source code. To promote cross-platform independence (which has consumed inordinate human resources in the present Greenstone system) and facilitate the dynamic loading of services and other modules, the implementation uses Java.

Following this we describe four very different examples built using the new design. The first demonstrates backwards compatibility, and, in addition, operates within a distributed environment. The second augments text at display time, using an established software tool for text mining developed elsewhere, to show how functionality can be enriched through the introduction of a new service. The third is a map-based digital library that introduces two new services to support geographic functionality and, while still accessed through a web-based interface, is highly tailored to the specific domain of its source documents. The fourth is a radically different interface for the interactive viewing of search results as a hierarchically organized cluster of documents, and illustrates the versatility of the design in coping with fine-grained interaction. We conclude with a commentary on how the design fits existing interoperability frameworks, from which many valuable lessons have been learned.

1.1 Background

Over the years the Greenstone digital library software has been employed by many users internationally to develop a wide variety of digital libraries. In addition to operation over the Web, an early application, and still a major one, is collections of humanitarian information in the form of CD-ROMs that run on any Windows computer (including Windows 3.1). There are now over two dozen of these collections, and they have been distributed widely by the United Nations and other non government agencies [9].

Many other styles of collection have been developed under Greenstone. They range from numerous personal collections based around common document formats such as E-mail, photographs, Word, and PDF documents through to large-scale bibliographic catalogs such as the BBC radio and TV archives. There are mixed media collections involving text, images and audio drawn from data such as historic newspapers and oral history. Several demonstration music collections support direct content-based retrieval through “query by humming.” Many customized and branded user interfaces exist, such as the New York Botanical Gardens, and many international collections in local languages created by institutions in China, India, Croatia, Russia and Israel.

The software is distributed in source form, and for convenience binaries are available for Windows, Linux and MacOS X. The user interface is available in 30 languages. A recent addition is a graphical interface for collection design and construction [2] which is also multilingual. A portfolio of demonstration collections is available at *www.nzdl.org*, and selected example collections built internationally can be found at *www.greenstone.org*.

1.2 Weaknesses

Many experimental interfaces have been built for Greenstone, some of which make use of a CORBA-based protocol to support distributed client-server interaction. These include a Venn diagram tool for formulating Boolean queries graphically, and a bibliographic visualization tool that plots matching citations on an *x*-*y* grid based on publication year and ranked relevance score to the query terms [1]. However, while perfectly functional, some of these variants are implemented inelegantly, exposing limitations caused by certain aspects of the design. For instance, the immutable nature of the index files generated during the building process makes incremental adjustments to a collection expensive. Another hindrance is the low level of functional customization supported by the runtime system. Although minor presentation tweaks are easy, more extensive changes involve modifying and recompiling the source code. The strongly typed nature of the CORBA protocol was also found to be overly restrictive in many practical situations.

1.3 Requirements

From these and other considerations arose the following requirements for an improved design.

Backwards compatibility. Naturally we wish to retain the existing system's strengths. This is accomplished by ensuring that the new design is backward compatible, which has the added benefit of providing existing developers and users with an easy migration path.

Levels of customization. To match the different categories of people involved in constructing digital libraries, different levels of customization are required. For instance, a content developer may wish to include source documents in a new format; a collection editor may seek influence over diverse issues of presentation.

Software modularity. To facilitate development and long-term management of the software, code modularization—a mantra of any software engineering approach—is essential. This is promoted by adopting off-the-shelf technology such as a database system, indexing tools, and page rendering software; and by the use of standards.

Service based. Basing a digital library around a set of services is another way to accomplish modularity—in this case, modularity of function.

Distributed architecture. A rich digital library infrastructure can only be supported by a distributed architecture, and the addition of an open protocol helps to foster interoperability.

Future compatibility. Libraries are long-term institutions with a mandate for preservation, and it is essential that old collections can be presented by future versions of the system. This is a more ambitious requirement than mere extensibility which, although an admirable quality in any design, does not necessarily ensure that future versions can safely interact with current ones.

Dynamic. Many aspects of the library should be dynamic, for example *content*, whereby documents and metadata can be added, revised and removed while a repository remains on-line, and *configuration*, where presentational issues can be adjusted and services added at runtime.

Integrated documentation. Large-scale software systems such as digital libraries benefit immensely from the use of an integrated documentation system.

Self-describing modules. This goes a stage further than the previous item: modules describe themselves in a machine-readable format so that other modules can interact with them without the need for explicit control.

Computer environment integration. A digital library should mesh well into a user's existing computer environment. Full integration makes a digital library become a seamless component of each user's work environment.

2 Software Design

The new software design has evolved from the requirements articulated above, our own experience of developing digital library systems, and studies of other open source software and research projects. This section provides an overview of modularity and inter-module communication, and then works through a simple, stand-alone example. The next section describes four very different examples built using the new design, to illustrate the general applicability of the approach.

2.1 Modularity

We decided that the best way to meet the challenges posed by the list of requirements was to reimplement Greenstone using a modular topology. In the design, a digital library manifests itself as a network of modules, and communication between them is expressed through an instantiation of XML. A mandatory requirement for a module—enforced through its base class definition—is that it handles "describe yourself" messages. What kinds of messages follow typically depend on the outcome of an initial "describe yourself". Modules have the ability to transform messages by applying an XSL Transform. This is a particularly useful mechanism for dynamically controlling levels of configurability—one of our requirements. Within the network of modules comprising a digital library, a set of services are defined that prescribe the functionality supported by that particular digital library configuration.

2.2 Communication

Modules communicate using synchronous request-response pairs. Fig. 1 shows an example XML exchange, where the TextQuery module in a collection called "demo" is asked to describe itself. It does this, providing information about the structure of the service as well as fragments of text to be used for display. These are returned in the language specified in the request. While the language used is structured and typed, it has been crafted in such a way that the information it contains can be open-ended. For example, in Fig. 1 the *paramList* structure allows other items to be included, such as whether stemming is on or off. In addition to supporting optional parameters, extensions can be introduced without

```
<request to='demo/TextQuery' lang='en' type='describe' uid='21'/>

<response from='demo/TextQuery' lang='en' type='describe'>
  <service name='TextQuery' type='query'>
    <displayItem name='name'>Search</displayItem>
    <displayItem name='description'>Full text search</displayItem>
    <displayItem name='submit'>Search</displayItem>
    <paramList>
      <param name='index' type='enum_single' default='stx'>
        <displayItem name="name">Index to search in</displayItem>
        <option name="dtx">
          <displayItem name="name">entire documents</displayItem>
        </option>
        <option name="stx">
          <displayItem name="name">chapters</displayItem>
        </option>
      </param>
      <param name='maxDocs' type='integer' default='10'>
        <displayItem name="name">Maximum hits to return</displayItem>
      </param>
      <param name="query" type="string">
        <displayItem name="name">Query string</displayItem>
      </param>
    </paramList>
  </service>
</response>
```

Fig. 1. A sample XML exchange as a request/response pair.

having to update the protocol's API, thereby avoiding the need to propagate code changes to the entire distributed network (which may require taking it off-line while the changes are made). For a more thorough technical description of the design, see the integrated documentation that is supplied with the Greenstone3 source code (available through SourceForge).

2.3 A simple stand-alone example

The simple stand-alone example shown in Fig. 2 encapsulates many of the design's key features. It comprises a "back end" server, termed a digital library *site* in our design, coupled to a "front end" that provides the user interface. The modules' names were chosen to emphasize the roles they play. The Receptionist's point of contact with the server is the MessageRouter (MR) module—all communication with the site occurs through this module. The configuration is designed to generalize to a distributed environment.

The digital library back end in Fig. 2 contains two collections, *demo* and *fao*, and a cluster of collection-formation services. *AddDocument* is a service that adds a document to a collection. *ImportCollection* imports into the system all documents associated with a collection, converting them where necessary from their original form. *BuildCollection* builds all indexes and browsing structures that are associated with a collection. *ActivateCollection* makes a newly-built collection active, so that it can be seen by digital library users. These services are all concerned with creating a digital library collection. Being related, they are grouped together into a "service cluster." In Fig. 2 the services just listed are accessed through the *CollectionFormation* service cluster module.

As far as the digital library user is concerned, a "collection" is a focused group of documents with a uniform means of access. For the system, it is a service cluster that groups a set of services that are related by the data they work on. For

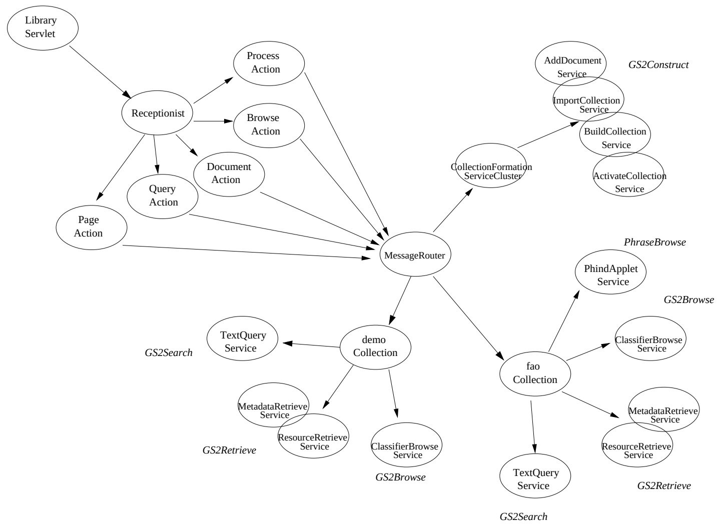


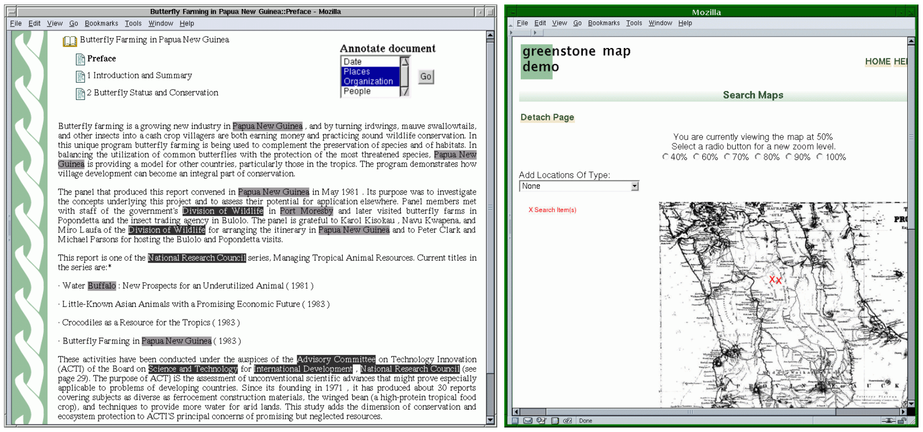
Fig. 2. A simple stand-alone site.

example, the *demo* collection towards the lower left of Fig. 2 contains three modules providing four services to the user. The modules are called “Search,” which provides a *TextQuery* service, “Retrieve,” which provides a document retrieval service *ResourceRetrieve* and a metadata retrieval service *MetadataRetrieve*, and “Browse,” which provides a metadata browsing service *ClassifierBrowse*.

The Web-based front end at the upper left of Fig. 2 centers around the Receptionist, which is the point of contact for the interface generator. A servlet (labeled “Library Servlet”) takes HTTP commands in the form of URLs and arguments and translates them into XML for the Receptionist. The Receptionist is capable of executing various different actions, each of which involves (usually) many calls to the digital library's MessageRouter (center of Fig. 2). The built-in ability for Receptionist modules to transform XML messages using XSLT is used—in conjunction with a style sheet—to generate the HTML that is finally presented to the user.

3 Examples

For pedagogical purposes, the example configuration in Fig. 2 is rudimentary. The front and back ends (receptionist and site server) are compiled together into a single executable process with a single MessageRouter handling all communication between them. However, the design supports a far richer infrastructure than this. MessageRouters have the capability to communicate across a distributed network with other MessageRouters and with Receptionists. Different implementations of the same service can be switched in and out to give a digital library site different algorithmic behavior (such as incremental building); and new services can be introduced and brought on-line, dynamically if necessary. The library can also have many different user interfaces.



Butterfly Farming in Papua New Guinea

Preface

1 Introduction and Summary

2 Butterfly Status and Conservation

Butterfly farming is a growing new industry in Papua New Guinea , and by turning irdwings, mauve swallowtails, and other insects into a cash crop villagers are both earning money and practicing sound wildlife conservation. In this unique program butterfly farming is being used to complement the preservation of species and of habitats. In balancing the utilization of common butterflies with the protection of the most threatened species, Papua New Guinea is providing a model for other countries, particularly those in the tropics. The program demonstrates how village development can become an integral part of conservation.

The panel that produced this report convened in Papua New Guinea in May 1981 . Its purpose was to investigate the concepts underlying this project and to assess their potential for application elsewhere. Panel members met with staff of the government's Division of Wildlife in Port Moresby and later visited butterfly farms in Popondetta and the insect trading agency in Bulolo. The panel is grateful to Karol Kisokau , Navu Kwapena, and Miro Laufa of the Division of Wildlife for arranging the itinerary in Papua New Guinea and to Peter Clark and Michael Parsons for hosting the Bulolo and Popondetta visits.

This report is one of the National Research Council series, Managing Tropical Animal Resources. Current titles in the series are:*

· Water Buffalo : New Prospects for an Underutilized Animal (1981)

· Little-Known Asian Animals with a Promising Economic Future (1983)

· Crocodiles as a Resource for the Tropics (1983)

· Butterfly Farming in Papua New Guinea (1983)

These activities have been conducted under the auspices of the Advisory Committee on Technology Innovation (ACTI) of the Board on Science and Technology for International Development , National Research Council (see page 29). The purpose of ACT} iS the assessment of unconventional scientific advances that might prove especially applicable to problems of developing countries. Since its founding in 1971 , it has produced about 30 reports covering subjects as diverse as ferrocement construction materials, the winged bean (a high-protein tropical food crop), and techniques to provide more water for arid lands. This study adds the dimension of conservation and ecosystem protection to ACTI'S principal concerns of promising but neglected resources.

You are currently viewing the map at 50%
Select a radio button for a new zoom level.
○ 40% ○ 60% ○ 70% ○ 80% ○ 90% ○ 100%

(a) (b)

Fig. 3. Additional services to augment run-time functionally (a) text augmentation of place names; (b) a novel map-based interface.

We now describe four more interesting scenarios. The first emulates "classic" Greenstone to demonstrate backwards compatibility, with a distributed configuration a straightforward addition. The second augments text at display time, to show how functionality can be enriched through the introduction of a new service. The third and fourth utilize novel interfaces to enrich the interaction: the former is specialized for geographic maps, and the latter for text based collections.

3.1 Classic Greenstone, with collections on remote sites

To provide backwards compatibility, a set of Greenstone3 services have been implemented that access index and database files generated by a Greenstone2 system. Indeed we have already seen these in Fig. 2. Moreover, using Cascading Style Sheets (CSS), the look and feel of the traditional interface has been matched one-for-one, so that the user's experience of the digital library remains the same. Of course with the more versatile design, more sophisticated internal structures can be built within the digital libary. For instance, because of the design decision that all communication be XML based, it is straightforward to arrange for the XML messages sent between modules to be seamlessly streamed across remote machines. For this we employ the Simple Object Access Protocol (SOAP), but any protocol that can process text can be used. To the user there is no disernable difference accessing collections this way. Backwards compatability is not the sole aim, however, and the structure allows increasingly interesting digital library designs—ones that do provide a distinctly different expierence for the user. We discuss three such examples next.

3.2 Dynamic text mining

The next example shows how a new Greenstone3 service can be provided that enhances documents for the reader, based upon the open source GATE system for text mining [5]. The motivation is that highlighting selected items in documents can make it easier for users to scan them for particular pieces of information. In this implementation, digital library documents, once located, are mined on the fly marking up entities such as keywords and place names before presenting them to the user [8].

Fig. 3(a) shows a section of a book on Butterfly Farming in Papua New Guinea (from the Humanity Development Library at *www.nzdl.org*) that the user is reading. The Annotate document menu at the top right calls the display-time text mining feature. When items on this menu are selected, text of the selected type is highlighted in the document displayed. In this case the user has selected Places and Organizations, and these are highlighted wherever they occur, in different colors (simulated by white-on-grey and black-on grey in the illustration).

The selectable items in Fig. 3(a) were identified and extracted by ANNIE, GATE's information extraction system. Whenever an annotation type is selected from the menu, Greenstone3 calls the information extraction module dynamically to identify items of that type. This incurs a delay that depends on the document's size but is usually short: the system processes text at the average rate of 15 Kb/sec on a typical workstation. In order to achieve this the software is loaded when the Greenstone3 server is executed and remains resident in memory thereafter, to avoid a start-up delay (of about 10 seconds) in which all the text processors are initialized and prepared for use.

In Greenstone3, GATE is encapsulated in a module as just another service. It takes messages with two parameters, the type of annotation and a document, and returns the document with relevant items marked up with XML tags. A style sheet in the server module chooses the colors in which the items are displayed.

In order to generate the menu in the figure, a "describe yourself" message is sent to the GATE module, which returns a list of the annotations that are available. Apart from this one module, the rest of the digital library system knows nothing of GATE. However, it does know about a general class of services, *Enrich*, that take a document and return the same document with some elements marked up. To add this text-mining ability to an existing Greenstone3 site dynamically the Java class files for the new service need to added to the correct folder on the DL site's file system, the site's configuration file updated to name the new service, and a "reconfigure" signal sent (which can be initiated through a web browser) to the DL site server.

3.3 Geographical map based interface

Fig. 3(b) is taken from a site configured to support map searching based on a small collection of New Zealand maps drawn between 1770 and 1953, part of our university library's special collections. It shows the result of searching for "Rangiriri"—a township in the North Island of New Zealand and, on a hill nearby of the same name, the scene of a battle in 1863 between Maori and British soldiers. Two crosses (colored red when viewed online) have been overlaid on the map showing the position of the township and hill, and can be seen slightly above the center of the map.

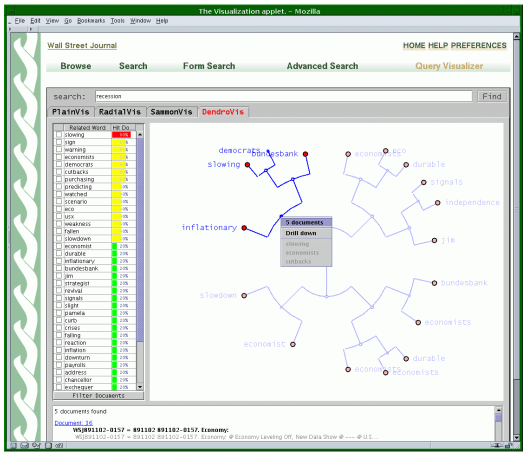


Fig. 4. Applet based Dendro visualization of clustered documents matching query.

The query interface and presentation of the results is similar to the traditional interface, except that against each matching item a thumbnail of the map containing the place name is given. Fig. 3(b) is the result of honing in on one such map appearing in the result set. Through the interface the user can change the level of magnification of the map and request that additional features such as cemeteries, bridges, and estuaries be marked.

This new functionality is provided by adding two new services, including geographic metadata with the collection's source documents (the maps), and a gazetteer of over 40,000 entries from Land Information New Zealand (LINZ). Searching is handled by the *MapQuery* service. For each place name query term that is located in the gazetteer, its longitude and latitude are checked against each map to see if it falls within its boundaries. Once all cross-referencing is complete, the list of maps with matching entries is sorted by the number of query terms on each map. Image manipulation and the results display are handled by the *MapRetrieve* service, which makes use of the open source ImageMagick toolkit for copying, annotating (such as placing a cross at an x, y location) and resizing images.

3.4 Cluster visualizations

Fig. 3 shows a web-based interface, but there may be other forms, ranging from standalone applications and applets that display documents in different ways to alert services that recognize when new information becomes available in one of the collections and formulate appropriate E-mail to users. Fig. 4 is one such instance: a Java applet that supports browsing result sets as document clusters through a multiviewed, interactive interface [3]. To function it needs the addition of a new build-time service, *VisConstruct*, which is responsible for computing additional statistics about the collection necessary to perform subsequent document clustering operations, and a sub-classing of the *Applet* service, tailored for the HTML necessary to embed the visualization applet within a web page.

Besides this, the visualization can interact with any text-based Greenstone collection that defines keyword metadata. The distinctly different interface behavior is achieved through fine-grained interaction with the existing *TextQuery* and *MetadataRetrieve* services using SOAP.

Derived from the query term entered, there are four views in the visualization. The first is the standard result list format; the other three are based around a shared model of hierarchically clustered documents. In Fig. 4 the user has entered the query term "recession" to a collection based on the Wall Street Journal and is viewing the result as a Dendro map—the circular arranged tree displayed in the main panel—which contains words like *slowing*, *slowdown* and *inflationary* around the circumference.

The size of nodes in the tree represents the number of documents clustered at that point in the hierarchy, and for reasons of space only 5 levels of the tree are shown at a time. Clicking on a node shows a popup menu with the precise number of documents in the cluster and the option of "drilling down" as shown in the figure. The consequence of choosing this option is for the tree to be redrawn with the selected node at the center of the map, thereby exposing more of the detail in that part of the hierarchy. While all this is going on, the left-panel displays the keywords relevant to the selected cluster, and the bottom panel lists the documents involved. Within the left panel, keywords can be activated or deactivated and the bottom panel dynamically filtered to reflect the changes.

4 Interoperability frameworks

This modular, service-based and dynamically configurable approach to digital library architecture can be found in other designs and systems. Two key examples are the Extended Open Archives Initiative protocol (XOAI) proposed by Suleman and Fox [6] and the OpenDLib system of Castelli and Pagano [4].

The XOAI protocol, based upon XML and related technologies, builds upon the basic OAI Protocol for Metadata Harvesting. It is a modular protocol which represents each service as a different type of OAI request. This service-based approach is readily seen in Greenstone3's protocol, where each service has its own request and response format. Like OAI and XOAI, Greenstone3 uses XML for representing requests and responses; providing a gateway between Greenstone3 and XOAI servers and clients should be readily achievable.

The OpenDLib system also takes a modular approach to service communication and library construction. Many of its features are similar to those of XOAI and Greenstone3, and the handling of requests to the library map as readily as for XOAI. However, OpenDLib is centered upon a "manager" service that coordinates services, responding to the introduction of new services and alterations to existing ones. Coordination is achieved through a communication protocol between the manager and each service. Individual services, including each library in a federation, acquire data from the manager. The manager does not itself control or direct requests for searches, documents, and so on but instead coordinates between services to ensure consistency. Greenstone3 provides many of these features—for instance, service description requests—without requiring a centralized manager.

The similarity of protocols between Greenstone3, XOAI and OpenDLib echoes experience with the previous generation of DL protocols and augers well for interoperability. Greenstone's widespread adoption and free availability allows others to adopt it as a platform for experimenting with the service-centered design approach. It facilitates the addition of new digital library features which would have been hampered by the less open structures of Greenstone2 and its contemporaries. Greenstone3 retains and extends the lightweight configurability of its predecessor without requiring centralized management processes.

5 Conclusions

This paper has presented a new digital library design which is significantly different from the present Greenstone, but strongly rooted in past success. To meet a broad range of requirements a modular based topology is utilized to provide a flexible infrastructure. Written in Java to promote portability, dynamic loading of objects and internationalization, modules communicate by streaming XML messages between each other. Using SOAP this communication can be distributed across a network. All modules have the ability to describe themselves in a machine readable form, and to apply an XSLT to transform messages. The latter is instrumental in providing different levels of configurability, an important ability given the different types of people involved in the life-cycle of a digital library.

Several working examples have been given to convey the general applicability of the design. The first example demonstrated backwards compatibility to collections built with Greenstone2, and satisfies another of the identified requirements: to help minimize the migration path of existing developers and users. It also demonstrated the design functioning in a distributed environment. The second example augmented an existing DL site with text mining capabilities with the introduction of a new service. The third and fourth examples illustrated substantially different user interfaces: one specialized for maps that required two extra services to provide the necessary geographical functionality; the other provided an interactive visualization to search results through a Java applet that was applicable to a wider realm of text-based collections.

Numerous aspects of the design contribute to its dynamic configurability. The "describe yourself" feature allows receptionists to adapt to the facilities that a server presents. Through XSLT a corpus editor can exert significant control over the function and appearance of a digital library such as modifying the structure of a generated page, for example adding a transform that sorts the query results alphabetically by title rather than ranked score. Alternatively they may reformat the results according to OAI syntax in preparation for exporting. Previously such changes required editing the source code and restarting the digital library software after recompilation.

Implementing the design in Java has promoted dynamic attributes in the system, particularly through the ease in which modules can be loaded at runtime.

The re-reading of configuration files is also straightforward to arrange and adds dynamic abilities that a digital library administrator can take advantage of. In addition to integrating the documentation with the software using JavaDoc and other techniques, usability of the software also benefits from the development of self-documenting collections that both demonstrate and describe particular aspects of design. The list of requirements is ambitious, however, and not all of them have been proven yet. For example, while none of the sample implementations demonstrate computer environment integration, the design is capable of supporting the idea through specialized receptionists. For future compatibility, more time is needed to establish if the design successfully meets such a criteria.

Acknowledgments

We gratefully acknowledge the stimulating research environment provided by the New Zealand Digital Library Project, and in particular Michael Dewsnip; Stefan Ruger and Daniel Heesch of Imperial College London; Valentin Tablan of the University of Sheffield, all of whom contributed to certain aspects of this work. This research is supported by the New Zealand New Economy Research Fund.

References

1. D. Bainbridge, G. Buchanan, J. McPherson, S. Jones, M. Mahoui, and I. H. Witten. Greenstone: a platform for distributed digital library applications. In *Proceedings of the European Conference on Digital Libraries*, pages 137–148, Darmstadt, Germany, September 2001.
2. D. Bainbridge, J. Thompson, and I. H. Witten. Assembling and enriching digital library collections. In *Proc. of the third ACM and IEEE joint conference on Digital Libraries*, pages 323–334, Houston, Texas, 2003.
3. M. Carey, D. C. Heesch, and S. M. Rüger. Info navigator: A visualization tool for document searching and browsing. In *Proc. of the Intl. Conf. on Distributed Multimedia Systems (DMS)*, September 2003.
4. D. Castelli and P. Pagan. A system for building expandable digital libraries. In *Proc. of the third ACM and IEEE joint conference on Digital Libraries*, pages 335–345, Houston, Texas, 2003.
5. H. Cunningham. GATE, a general architecture for text engineering. *Computers and the Humanities*, 36:223–254, 2002.
6. H. Suleman and E. A. Fox. Designing protocols in support of digital library componentization. In *Proceedings of the European Conference on Digital Libraries*, pages 568–582, Rome, Italy, 2002.
7. I. H. Witten and D. Bainbridge. *How to build a digital library.* Morgan Kaufmann, San Francisco, 2003.
8. I. H. Witten, K. J. Don, M. Dewsnip, and V. Tablan. Text mining in a digital library. *Journal of Digital Libraries*, 2003 (in press).
9. I. H. Witten, M. Loots, M. Fernandez-Trujillo, and D. Bainbridge. The promise of digital libraries in developing countries. *The Electronic Library*, 20(1):7–13, 2002.